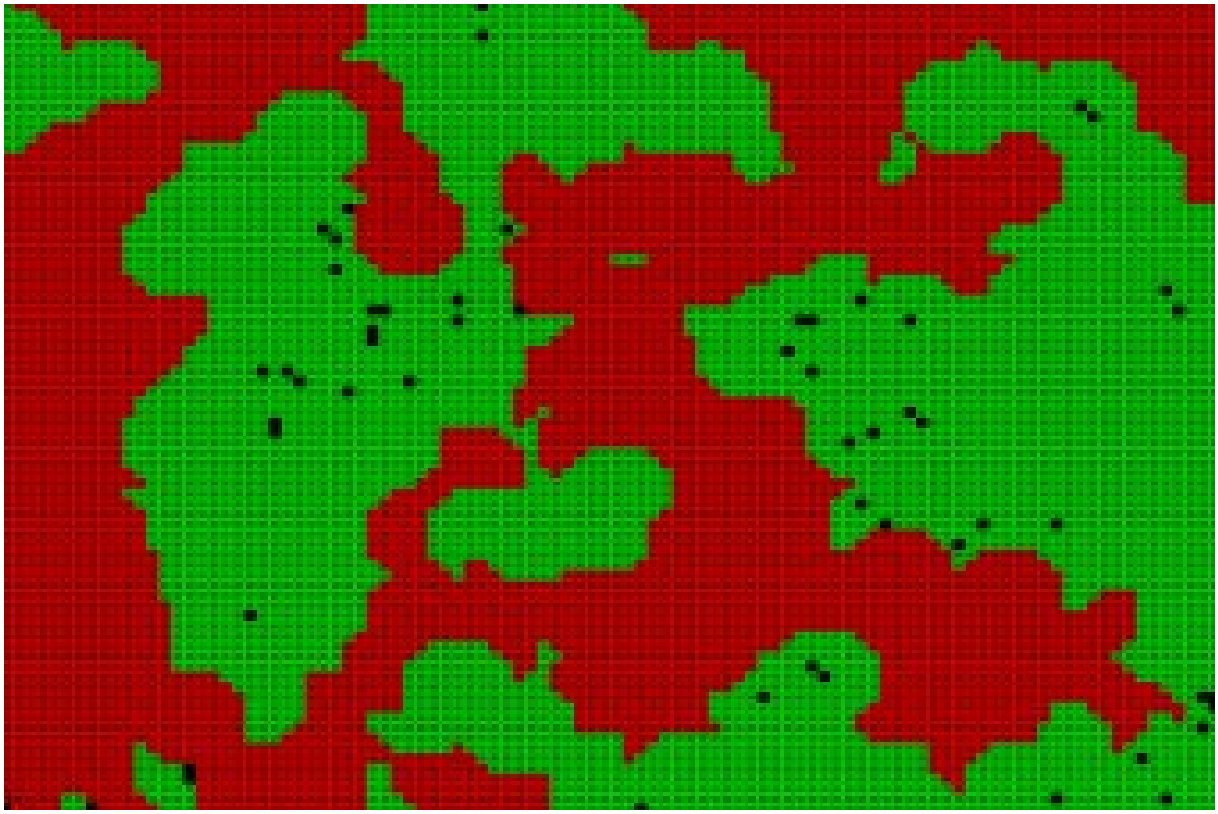
FACHHOCHSCHULE HANNOVER

Fakultät IV - Wirtschaft und Informatik
Abteilung Informatik

DATA MINING VERFAHREN

Forschungsbericht



Prof. Dr. Werner M. Lechner

werner.lechner@fh-hannover.de

März 2008

Vorwort

Der technische Fortschritt in den letzten beiden Jahrzehnten ermöglicht das persistente Speichern gewaltiger Datenmengen, die in allen Lebensbereichen automatisch erfasst und dann in großen Datenbanken verwaltet werden. Diese hochdimensionalen Datenstrukturen enthalten wertvolle Informationen und aufschlussreiche Zusammenhänge, die einen Wettbewerbsvorteil bieten, wenn sie erschlossen werden könnten. Allerdings stellt die Exploration großer Datenmengen eine enorme Herausforderung für mathematische und statistische Algorithmen dar.

Der folgende Bericht dokumentiert die im Sommersemester 2007 im Rahmen eines Forschungssemesters erzielten Ergebnisse zur intelligenten Analyse großer Datenmengen. Die grundlegende Aufgabe von Data Mining besteht darin, mittels geeigneter Algorithmen aus rohen Datensätzen wirtschaftlich oder technisch interessante Zusammenhänge zu extrahieren und daraus Wissen über den vorliegenden Prozess zu gewinnen. Dieses Wissen dient dann als Grundlage für entsprechende Aktionen und ermöglicht das interaktive Eingreifen zur Optimierung des Prozesses (siehe Abb.: 1).

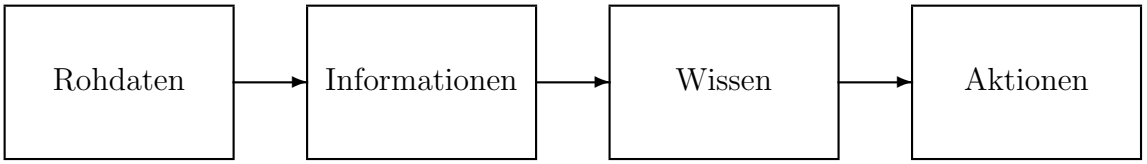


Abb.: 1: Das Grundprinzip von Data Mining

Für die Bereitstellung einer Vielzahl von Datensätzen gilt mein Dank der Firma prudsys AG sowie allen anderen Veranstaltern des jährlich stattfindenden Data Mining Cups. Bei diesen Wettbewerbsdaten handelt es sich um reale Daten aus der betrieblichen Praxis unterschiedlicher Unternehmen. Daher sind diese Daten hervorragend geeignet, um die Leistungsfähigkeit der in diesem Forschungsbericht beschriebenen Data Mining Methoden zu testen.

Hannover, März 2008 Prof. Dr. Werner Lechner

Das Titelbild auf der ersten Seite des Forschungsberichts stellt die „Feature Map“ eines neuronalen Kohonen-Netzes dar. Die roten bzw. grünen Bereiche visualisieren die beiden Klassen einer Datei, die aus 8000 protokollierten Datensätzen einer speziellen eBay-Auktion besteht. Jede einzelne Auktion entspricht einem Element in der Graphik. Die schwarzen Bereiche markieren jene Datensätze, welche der nicht abgeschlossene Trainingsalgorithmus einer falschen Klasse zuordnet. Die Datei stammt aus dem Data Mining Cup 2006 und dient als Datenbasis zur Vorhersage erfolgreicher eBay-Auktionen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Viele Firmen verfügen über riesige Datenbestände (Abb. 1.1, linkes Bild). Typische Beispiele dafür finden sich im Versandhandel, bei Versicherungen, bei Autoherstellern und bei Auktionshäusern. In diesen Datenbeständen verstecken sich interessante Zusammenhänge, die sich durch die Anwendung geeigneter mathematischer Algorithmen aufspüren lassen. Stellt man sich die Datenbestände wie ein Gebirgsmassiv an Informationen vor, so liegt die Vorstellung nahe, in diesen Berg mittels mathematischer Methoden tiefe Stollen zu treiben, um damit wirtschaftlich oder technisch wertvolle Informationen wie „Edelmetalle“ (Abb. 1.1, rechtes Bild) zu gewinnen. Durch die Umsetzung der aus der Datenbank gewonnenen Erkenntnisse gelingt es dann Marketingstrategien zu verbessern, die Qualität einer technischen Produktion zu erhöhen und ganz allgemein Wettbewerbsvorteile für ein Unternehmen zu erschließen.

Bei der Auswahl leistungsfähiger mathematischer Verfahren treten außer den statistischen Methoden verstärkt die Techniken der künstlichen Intelligenz in den Vordergrund. Neuronale Netze erkennen in einem Datenbestand unbekannte Zusammenhänge und ermöglichen deren anschauliche Visualisierung, z.B. in einer Featuremap (Abb. 1.1, mittleres Bild).

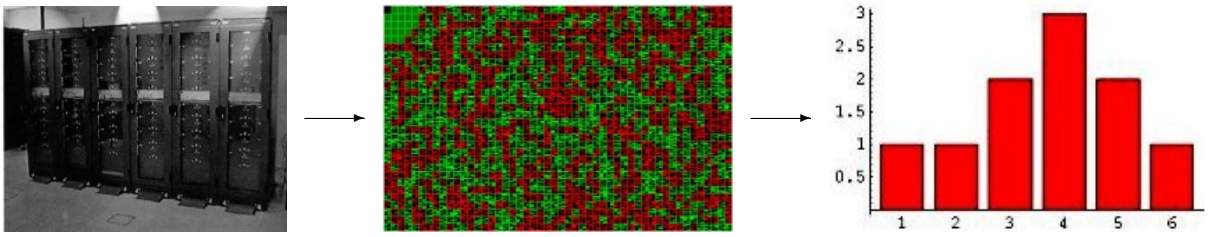


Abb.: 1.1: Das Grundprinzip von Data Mining Techniken

Künstliche neuronale Netze lösen seit ca. 20 Jahren eine Vielfalt praktischer Aufgaben in technischen, käufmännischen und medizinischen Anwendungsgebieten. Im Anhang finden sich unter dem Punkt 7.2.3 eine Reihe interessanter Beispiele, wobei sich der Verfasser dieses Forschungsberichts in den letzten Jahren selbst intensiv mit den folgenden Anwendungen neuronaler Netze befasst hat:

- Treibstoffminimierung bei Verkehrsflugzeugen [29]
- Intelligente autonome Steuerung von Robotern [30]
- Automatische Erkennung von Personen [32]

In den letzten Jahren richtet sich die Aufmerksamkeit vieler Forschungsarbeiten auf die neuartigen „Support Vektor Maschinen“ [17], , die von Vladimir Vapnik (Universität Leu-

ven, 1995) entwickelt wurden und die bei speziellen Data Mining Aufgaben eine faszinierend hohe Leistungsfähigkeit demonstrieren. Vor allem belgischen Forschern verbesserten die entsprechenden Algorithmen und schließlich gelang ihnen die Entwicklung einer stabilen und geschlossenen Berechnung der erforderlichen Koeffizienten auf der Basis von Least-Square Regressionsverfahren [23].

Data Mining Methoden stellen in der Regel hohe Anforderungen an die Rechenzeit. Daher fanden sie erst mit der Entwicklung schneller Computer ihren weltweiten Durchbruch. Stichworte in diesem Zusammenhang sind: On-Line Analytic Processing (OLAP), Knowledge Discovery in Databases (KDD), Web Mining, Text Mining, Statistical Exploratory Methods und Exploratory Data Analysis. Dabei bestimmt der Charakter der Aufgabenstellung die Wahl bzw. die optimale Kombination der geeigneten Verfahren.

Die folgende Auflistung beschreibt die vier wichtigsten Data Mining Methoden, die einzeln oder auch kombiniert eingesetzt, über ein enormes Potenzial beim Aufspüren versteckter Zusammenhänge verfügen:

- **Assoziationsmethoden**, die häufig bei der Warenkorb-Analyse Anwendung finden, berechnen die Wahrscheinlichkeiten für das gleichzeitige Auftreten von Ereignissen. Z.B. lässt sich aus Verkaufsdaten berechnen, mit welcher Wahrscheinlichkeit ein Kunde, der bereits die Artikel (A,B,C) erworben hat, sich auch noch für den Artikel D entscheiden wird. Eine spezielle Werbeaktion könnte dann das gesamte Sortiment (A,B,C,D) im Verkauf unterstützen.

- **Klassifizierungsmethoden** ordnen einem neu erfassten Datensatz eine bestimmte Klasse zu. Typische Klassen sind z.B. die Klassen der guten und der schlechten Kunden oder medizinische Diagnoseklassen. Die Berechnung der Algorithmen basiert auf Trainingsdatensätzen, deren Klassen bekannt sind. Anschauliches Beispiel dafür könnte die Klassifizierung von Studierenden nach ihren Personaldaten sein. Schreibt sich ein Studierender neu ein, dann liefert ein entsprechender Algorithmus die Zuordnung zu der vorab definierten Klasse der erfolgreichen oder der nicht erfolgreichen Studierenden. Klassifizierungsmethoden setzen bei den Trainingsdaten grundsätzlich die Kenntnis der Klassen voraus und erfordern daher überwachte Lernalgorithmen.

- **Clusterbildung** nennt man jene Methoden, die für einen Datenbestand mit unbekannten Klassen die gesuchten Klassen ermitteln. Für die Clusterbildung stehen keine Kriterien zur Verfügung, d.h. es handelt sich um nicht überwachtes Lernen. Für das obige Beispiel der Klassifizierung der Studierenden einer Hochschule bedeutet dies, dass Clusteralgorithmen die persönlichen Daten aller Studierenden automatisch und ohne Vorgabe von Regeln nach Klassen strukturieren. Somit setzt jede Klassifizierung eine vorher durchgeführte manuelle oder automatische Clusterbildung voraus.

- **Zeitreihenanalysen** befassen sich mit der mathematischen Analyse Zeitverläufen. Basierend auf Regressionsverfahren oder auf künstlichen neuronalen Netzen lassen sich die erforderlichen Systemparameter berechnen und dann weitere Datensätze extrapolieren. Der laufende Vergleich prognostizierter Datensätze mit den tatsächlich aufgetretenen Datensätzen ermöglicht die Bestimmung eines Integritywertes, der bei exakter Vorhersage den Wert von 100% annimmt und bei geringer Integrity ein Update der adaptiven Systemparameter erfordert. Typische Anwendungen finden sich im Bereich der Prognose von Aktienkursen und der Vorhersage von Klimadaten.

Data Mining Methoden bieten sich an, wenn eine nahezu unüberschaubare Menge gespeicherter Daten vorliegt. Man könnte dann „den Computer bitten, sich diese Daten anzusehen und zu prüfen, ob ihm dabei etwas auffällt.“ Die rohen Datenbestände enthalten nämlich wertvolle Informationen über zunächst unbekannte Zusammenhänge zwischen den Datensätzen.

Die auf dem Markt erhältlichen Data Mining Produkte sind häufig Teil eines größeren Softwarepakets, denn zusätzlich zur eigentlichen Anwendung der Data Mining Algorithmen sind die folgende Arbeiten auszuführen:

- Auswahl von Datenbanken, die mit der gestellten Aufgabe in einem kausalen Zusammenhang stehen.
- Auswahl relevanter Datensätze.
- Eliminieren von fehlerhaften, mehrfach vorhandenen und widersprüchlichen Datensätzen innerhalb der ausgewählten Datensätze.
- Skalieren der Attributswerte auf einen Wertebereich von z.B. ± 100.
- Aufteilen der skalierten Datensätze in eine Trainingsmenge zur Berechnung der Parameter des entsprechenden Data Mining Verfahrens und in eine Testmenge, die zum Nachweis der praktischen Leistungsfähigkeit der Algorithmen dient. Damit stellt mach sicher, dass der Algorithmus die Trainingsdaten nicht nur auswendig lernt und dann zur Generalisierung bei den Anwendungsdaten versagt.
- Festlegung der konkret anzuwendenden Data Mining Methode bzw. deren optimale Kombinationen.
- Testläufe zur Optimierung des eingesetzten Data Mining Verfahrens.
- „Cross Validation“ durch ein zufälliges Variieren der Aufteilung der Datensätze in die Trainings- und Testmenge.

Die folgende Aufzählung nennt typische und ganz aktuelle Beispiele für den erfolgreichen Einsatz von Data Mining.

- Auswertung von Statistiken (Unfallstatistiken, medizinische Statistiken) [31]
- Warenkorbanalyse im Versandhandel und bei Supermärkten [35]
- Betrugsentdeckung bei Geldanlagen oder bei der Auftragserteilung [33]
- Marktanalyse zur Gewinnung von Neukunden [34]
- Customer Relationship Managment (CRM)[39]
- Vorhersage von Geschäftsdaten
  (Aktienkurse, Umsatzentwicklung, Preisentwicklung, usw.)[37]

War bisher im Zusammenhang mit Data Mining Verfahren immer nur bildlich von der Suche nach der Goldader „Gold Nugget“ in einem großen Berg aus Informationen die Rede, lassen sich umgekehrt auch die „Black Nuggets“ entdecken, womit verbotene Geldanlagen, Bestellung von Waren ohne Bezahlabsicht und ähnliche Aktivitäten gemeint sind.

Im kaufmännischen Bereich bilden Data Mining Verfahren eine Untergruppe des Hauptbegriffes der „Bussiness Intelligence (BI)“. Durch die Möglichkeiten des Web 2.0 entstehen völlig neuartige interaktive und auch rein virtuelle Geschäftsmodelle, bei denen Data Mining Verfahren die wesentliche Komponente darstellen.

Bei den Verfahren des „Competitive Intelligence (CI)“ sammelt man gezielt Informationen vergleichbarer Produkte von den Web-Seiten konkurrierender Firmen ein. Ein zunächst leerer Datenbestand füllt sich dabei automatisch mit den Inhalten der geparsten Web-Seiten und entsprechende Data Mining Verfahren liefern durch „Text Data Mining“ und durch die Auswertung dieser Datenbestände die gewünschten Reports.

Allgemein dient „Text Data Mining“ zum Auslesen relevanter Informationen z.B. aus einer Web-Seite oder einer E-Mail, wobei die gewünschten Daten in einer unstrukturierten Form vorliegen und geeignete Data Mining Algorithmen die Daten in eine strukturierte Form umwandeln.

Auf dem Software-Markt sind zahlreiche Data Mining Tools erhältlich, die als einzelnes Programmpaket (Clementine), als WEB-basierter Service (SOA) oder als Teil eines größeren Software-Systems (SPSS, Oracle-Datenbank, Microsoft SQL Server) vorliegen.

Eine Vielfalt von Open Source Produkten, für die häufig auch eine kommerzielle Version existiert, ergänzt diese Palette:

- WEKA
- Rapid Miner
- KNIME.

In [38] vergleicht der Autor die bekanntesten Open Source Produkte mit dem rein kommerziellen Clementine-System und empfiehlt das kostenlose RapidMiner-Paket. Allerdings bezieht sich diese Aussage auf das Jahr 2007. Da laufend neue bzw. verbesserte Open Source Tools freigegeben werden, sollte der Anwender prüfen, welches Tool für seine Aufgaben am besten geeignet ist.

Kapitel 2

Assoziation

2.1 Warenkorb-Analyse

Der Begriff „Warenkorb“ kennzeichnet anschaulich die Auswahl einer Anzahl von M Produktengruppen aus einem Angebot von N Einzelprodukten, wobei jedes Produkt nur einmal gewählt werden kann und somit ein Warenkorb maximal M=N Produkte enthält. Ein Produkt kann einen konkreten Gegenstand umfassen, wie z.B. eine Sonnencreme, einen Weihnachtsbaum, usw. , jedoch z.B. auch die Jahreszeit, die Wetterlage usw.. Die Analyse einer großen Anzahl von Warenkörben ermöglicht die Generierung von Regeln, wie z.B. wer die Bücher a und b kauft, erwirbt auch mit einer bestimmten Wahrscheinlichkeit die Bücher c und d.

Ein gefüllter Warenkorb enthält viel mehr Informationen über den Käufer, als auf den ersten Blick erkennbar ist.

1. Relative Häufigkeit für den Verkauf eines einzelnen Produktes
2. Relative Häufigkeit für den Verkauf einer bestimmten Gruppe von Produkten
3. Assoziationsregeln der Form „Wer a und b kauft, kauft auch c und d“
4. Statistische Masszahlen für das Eintreffen der Assoziationsregeln

Der Begriff Warenkorb kennzeichnet anschaulich eine Ansammlung von Einzelprodukten, die mathematisch gesehen einer Menge entsprechen. Das mehrfache Auftreten eines bestimmten Einzelproduktes im Warenkorb wird dabei grundsätzlich nur einmal berücksichtigt. Es kommt somit auf die logische Aussage an, ob ein Einzelprodukt überhaupt oder gar nicht in der Menge auftritt. Mehrere Einzelprodukte lassen sich zu einer Produktkombination zusammenfassen. So sind z.B. bei drei Einzelprodukten (a,b,c) insgesamt acht unterschiedliche Kombinationen möglich:

$$(a, b, c) \quad \longrightarrow \quad () \quad (a) \quad (b) \quad (c) \quad (ab) \quad (ac) \quad (bc) \quad (abc)$$

Die erste Kombination entspricht einem leeren Warenkorb und wird häufig nicht weiter beachtet. Die Anzahl der maximal möglichen Kombinationen M_{max} entspricht gemäß der Gl.: 2.1 der Zweierpotenz von N:

$$M_{max} = 2^N \tag{2.1}$$

Warenkorb-Beispiel I

Eine Firma bietet drei Produkte a,b und c an. Der Inhalt von vier Warenkörben sei wie folgt angenommen (Tab.: 2.1):

Warenkorb	Inhalt
1	a
2	b
3	ab
4	abc

Tab.: 2.1: Warenkorb-Beispiel I

Daraus berechnet sich die Tabelle 2.2 mit den relativen Häufigkeiten.

Produktkombination	Anzahl	relative Häufigkeit je Produktkombination
a	3	$\frac{3}{4}$ bzw. 75%
b	3	$\frac{3}{4}$ bzw. 75%
c	1	$\frac{1}{4}$ bzw. 25%
ab	2	$\frac{2}{4}$ bzw. 50%
ac	1	$\frac{1}{4}$ bzw. 25%
bc	1	$\frac{1}{4}$ bzw. 25%
abc	1	$\frac{1}{4}$ bzw. 25%

Tab.: 2.2: Relative Häufigkeit der Produktkombinationen für Warenkorb-Beispiel I

Die Tab.: 2.3 enthält für alle sieben Produktkombinationen die Quotienten der absoluten Häufigkeiten der in einer bestimmten Produktkombination enthaltenen Unterkombinationen. So besteht z.B. die Produktgruppe (ab) aus den Unterkombinationen (a) und (b). Da (ab) zweimal auftrat und (a) bzw. (b) insgesamt dreimal in den Warenkörben lag, ergeben sich die entsprechenden Quotienten jeweils zu $\frac{2}{3}$.

	a	*b*	*c*	*ab*	*ac*	*bc*	*abc*
a	-	$\frac{2}{3}$	$\frac{1}{3}$	-	-	$\frac{1}{3}$	-
b	$\frac{2}{3}$	-	$\frac{1}{3}$	-	$\frac{1}{3}$	-	-
c	$\frac{1}{1}$	$\frac{1}{1}$	-	$\frac{1}{1}$	-	-	-
ab	-	-	$\frac{1}{2}$	-	-	-	-
ac	-	$\frac{1}{1}$	-	-	-	-	-
bc	$\frac{1}{1}$	-	-	-	-	-	-
abc	-	-	-	-	-	-	-

Tab.: 2.3: Absolute Häufigkeit von Produktkombinationen für Warenkorb-Beispiel I“

Ein Blick auf die absoluten Häufigkeiten der Tab.: 2.3 sagt z.B. aus: Bestellt ein Kunde das Produkt (c) (siehe Zeile c), dass er dann die Produkte (a), (b) und ab mit einer relativen Häufigkeit von 1/1 bzw. 100% ordert, was in den Spalten a, b und ab zu erkennen ist. Ein Blick auf die vier Warenkörbe bestätigt dieses Ergebnis: Das Produkt (c) wurde nur als Produktkombination (abc) erworben, d.h. wer (c) kauft, erwirbt immer auch die Produkte (a) und (b) sowie die Produktkombination (ab). Selbstverständlich erfordert diese Aussagen eine statistisch betrachtet ausreichend große Zahl von Warenkörben, damit die relativen Häufigkeiten mit hinreichender Genauigkeit auch als Wahrscheinlichkeiten interpretiert werden dürfen.

Nachvollziehbar ist auch der Wert von 1/2 bzw. 50% für das Produkt c, unter der Annahme(Prämisse), dass ein Produkt ab bereits im Warenkorb liegt. Wie die Inhalte des 3. und 4. Warenkorbes zeigen, taucht die Produktgruppe ab im 3. Warenkorb ohne und im 4. Warenkorb mit dem Produkt c auf. Daher kann man der Regel ab $\rightarrow$ c nur zu 50% vertrauen.

2.2 Support und Konfidenz

Die Begriffe Unterstützung und Vertrauen, die international als Support und Konfidenz bekannt sind, definieren die relativen Häufigkeiten für das Auftreten von Produkten bzw. Produktkombinationen bei einer ausreichend großen Zahl von Warenkörben. Erst bei einer genügend großen Zahl von Warenkörben geht gemäß den Gesetzen der Statistik die relative Häufigkeit in eine Wahrscheinlichkeit über und vertrauenswürdige Aussagen sind möglich. Einen Warenkorb bezeichnet man im Hinblick auf andere Anwendungsbereiche allgemein als Transaktion, das Element X als die Prämisse und das Element Y als Konklusion.

$$\boxed{\text{Support} \quad (X \to Y) = \frac{\text{relative Häufigkeit der Regel} \quad (X \to Y)}{\text{Anzahl der Transaktionen}}} \tag{2.2}$$

$$\boxed{\text{Konfidenz} \quad (X \to Y) = \frac{\text{relative Häufigkeit der Regel} \quad (X \to Y)}{\text{relative Häufigkeit der Prämisse}}} \tag{2.3}$$

Im Warenkorb-Beispiel I traten diese Begriffe bereits auf. Das Warenkorb-Beispiel II zeigt, wie sich die Ergebnisse aus dem Warenkorb-Beispiel I nun mittels der Begriffe Support und Konfidenz darstellen lassen.

Warenkorb-Beispiel II

Die Datenbasis stellen wieder die vier Warenkörbe mit den Inhalten (a), (b), (ab) und (abc) aus dem Warenkorb-Beispiel I dar.

Regelnummer	Regel X → Y	Support X	Support (X+Y)	Konfidenz X → Y
1	a → b	$\frac{3}{4}$ bzw. 75%	$\frac{2}{4}$ bzw. 50%	$\frac{2}{3}$ bzw. 66%
2	a → c	$\frac{3}{4}$ bzw. 75%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{3}$ bzw. 33%
3	ab → c	$\frac{2}{4}$ bzw. 50%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{2}$ bzw. 50%
4	ac → b	$\frac{1}{4}$ bzw. 25%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{1}$ bzw. 100%
5	b → a	$\frac{3}{4}$ bzw. 75%	$\frac{2}{4}$ bzw. 50%	$\frac{2}{3}$ bzw. 66%
6	b → c	$\frac{3}{4}$ bzw. 75%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{3}$ bzw. 33%
7	bc → a	$\frac{1}{4}$ bzw. 25%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{1}$ bzw. 100%
8	c → a	$\frac{1}{4}$ bzw. 25%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{1}$ bzw. 100%
9	c → b	$\frac{1}{4}$ bzw. 25%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{1}$ bzw. 100%
10	a → bc	$\frac{3}{4}$ bzw. 75%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{3}$ bzw. 33%
11	b → ac	$\frac{3}{4}$ bzw. 75%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{3}$ bzw. 33%
12	c → ab	$\frac{1}{4}$ bzw. 25%	$\frac{1}{4}$ bzw. 25%	$\frac{1}{1}$ bzw. 100%

Tab.: 2.4: Support und Konfidenz für das Warenkorb-Beispiel II

Die Regeln 4, 7, 8, 9 und 12 enthalten als Prämisse das Produkt (c), das nur in einem Warenkorb vorhanden war. Mit der Prämisse, dass ein Kunde das Produkt (c) erwirbt, besteht eine Wahrscheinlichkeit von 100% für den gleichzeitigen Kauf aller Produktkombinationen mit dem Produkt (c).

Interessant ist ein Blick auf die Regel 7. Obwohl bei den vorliegenden Warenkörben kein Kunde die Produktkombination (bc) einzeln erworben hat, besteht doch eine Wahrscheinlichkeit von 100% für den noch nicht eingetretenen Fall, dass ein Kunde beim Kauf von (bc) immer auch das Produkt (a) dazu erwirbt. Erinnert man den Kunden an diesen Zusammenhang, dann trägt dies zur Umsatzsteigerung bei.

Insgesamt folgen aus den drei Einzelprodukten (a), (b), (c) insgesamt sieben nicht leere Produktkombinationen und zwölf Regeln.

Die Berechnung des Supports und der Konfidenz erfordert die Bestimmung der Anzahl und der Zusammensetzung der Produktkombinationen sowie die Aufstellung der Regeln. Die Anzahl der Produktkombinationen hängt nur von der maximalen Anzahl der N Einzelprodukte ab, denn es liegt hier gemäß den Grundlagen der Kombinatorik eine Auswahl von k Elementen aus N Elementen ohne Berücksichtigung der Reihenfolge vor. Für diesen Fall gilt die Formel:

$$\boxed{\text{Anzahl der Produktgruppen mit k Einzelprodukten} = \binom{N}{k}} \tag{2.4}$$

Daraus ergibt sich dann die maximale Anzahl aller nicht leeren Produktgruppen M_{max} als binomische Summe, die für den Fall k=N gegen die Zweierpotenz konvergiert.

$$\boxed{\text{Anzahl aller Produktgruppen} \quad M_{max} = \sum_{k=1}^{N} \binom{N}{k} = 2^N - 1} \tag{2.5}$$

Mit einem Blick auf die Tab. 2.3 erkennt man, dass sich entlang der Diagonalen von links unten nach rechts oben genau (M-1) Regeln ergeben. Oberhalb dieser Diagonalen fehlt immer jene Regel, bei der sich die Produktgruppen auf sich selbst beziehen. Die Berechnung der Anzahl der maximal möglichen Regeln liefert für den Sonderfall $N = 3$ insgesamt $M = 7$ Produktgruppen und $R = 12$ Regeln. Mit steigender Anzahl von Produkten wachsen die Parameter M und R enorm an. Die Tab.: 2.5 veranschaulicht diesen Zusammenhang.

Einzelprodukte N	Anzahl der Produktkombinationen M	Anzahl der Regeln R
3	7	12
4	15	50
5	31	180
6	63	602
7	127	1.903
8	255	6.050
9	511	18.660
10	1.023	57.002
13	8.192	1.577.940

Tab.: 2.5: Anzahl von Produktgruppen und Regeln

Wie aus Tab.: 2.5 zu erkennen ist, steigt vor allem der Parameter R gewaltig an und führt zu untolerierbar langen Rechenzeiten. Durch das Weglassen von Regeln, die auf geringen Supportwerten basieren, begrenzt man bei praktischen Anwendungen den Rechen- und Speicheraufwand. Dies ist vor allem deshalb gerechtfertigt, weil die Division geringer, jedoch in etwa gleich großer Supportwerte, wie z.B. 10%/11%, zu hohen Konfidenzwerten führt, die wegen des geringen Supportwertes keine vertrauenswürdigen Aussagen ermöglichen. Kleine Supportwerte bedeuten eine geringe Wahrscheinlichkeit für das Auftreten der entsprechenden Produktkombination in einem Warenkorb. Daher sind Aussagen über innere Zusammenhänge dieser Produktkombination nicht zulässig.

Eine weitere Reduzierung der Anzahl der Regeln ergibt sich daraus, dass ein Kunde nicht alle möglichen N Produkte gleichzeitig kauft. Die Auswertung beginnt daher mit dem größten „Itemset“, d.h. dem am besten gefüllten Warenkorb. Außerdem lassen sich ganze Warengruppen, wie z.B. Food- und non Food - Artikel, als Einzelprodukte interpretieren, was die Zahl N ebenfalls drastisch reduziert. Diese und andere numerische Tricks ermöglichen die Handhabung der theoretisch sehr hohen Zahl von Produktkombinationen.

2.3 Generierung der Produktkombinationen

Bei einer konkreten Warenkorb-Analyse geht es darum den speziellen Inhalt eines einzelnen Warenkorbes nach darin enthaltenen Produktkombinationen zu untersuchen. Falls alle Einzelprodukte im Warenkorb liegen, treten alle M Produktkombinationen auf. Praktisch mögliche Produktkombinationen lassen sich vorab berechnen und speichern. Anschließend erfolgt für jeden einzelnen Warenkorb die Prüfung, welche speziellen Produktkombinationen tatsächlich im vorliegenden Warenkorb vorhanden sind.

Die Ermittlung der Produktgruppen geschieht durch die systematische Variation aller Produktkombinationen. Die Abb. 2.1 zeigt anschaulich für den Sonderfall N=4 die Bestimmung der Produktkombinationen. Es entstehen sechs Zweier- und vier Dreierkombinationen. Dazu addieren sich noch die vier Einzelprodukte sowie die Kombination (abcd). Insgesamt entstehen somit

$$M = 4 + 6 + 4 + 1 = 15$$

Produktkombinationen. Die 16. Produktkombination wäre übrigens der leere Warenkorb, der sich vorab aus dem Datensatz eliminieren lässt.

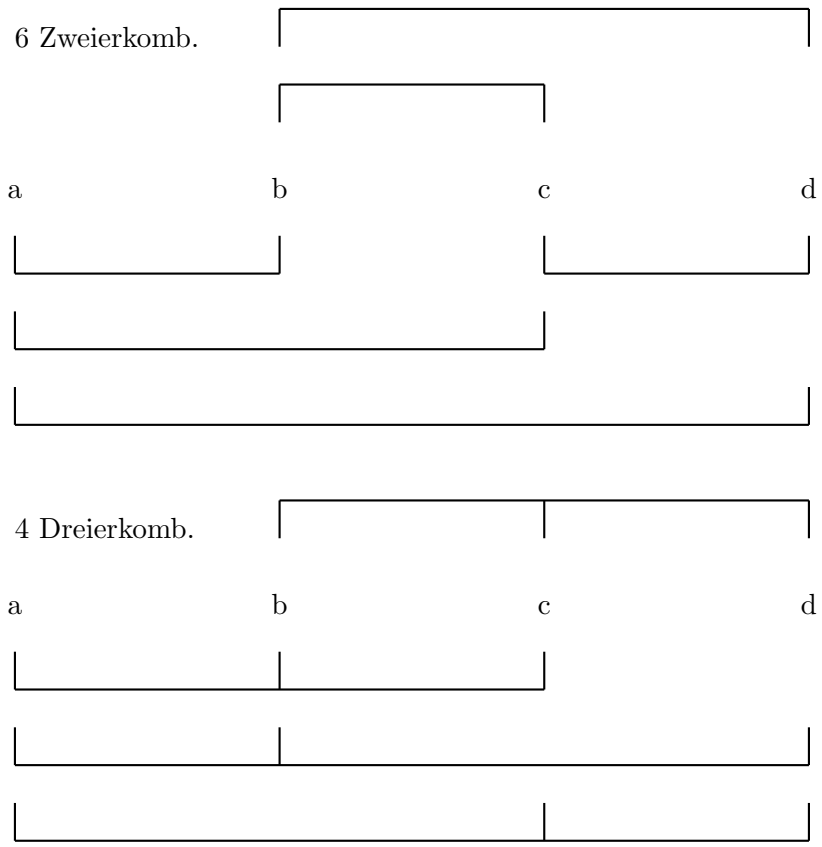


Abb.: 2.1: Bestimmung der Produktkombinationen für vier Einzelprodukte

Die Umsetzung der Bildung der Produktkombinationen in ein Computerprogramm folgt diesem Schema. Es entstehen geschachtelte Schleifen, wobei z.B. eine Produktkombination mit drei Produkten auch drei Schleifen benötigt und jeder Schleifenzähler mit dem momentanen Zählerstand der übergeordneten Schleife beginnt. Durch die Einführung logischer Warenkörbe im Zusammenhang mit einem Hashingverfahren, wie im folgenden Abschnitt beschrieben, vereinfacht sich die Berechnung der Produktkombinationen erheblich und es entstehen Computerprogramme mit einem realistischen Rechenzeit- und Speicherbedarf.

Beispiel: Java-Listing zur Generierung von möglichen Produktkombinationen

Das folgende JAVA-Listing zeigt den Programmcode einer Methode für einen Warenkorb N=4 und die entsprechende Ausgabe auf der Konsole. Es dient zur Veranschaulichung der Problematik bei der Generierung der Produktgruppen. Im Listing ist die Abhängigkeit der Schleifenzähler von der Anzahl der Einzelprodukte N gut erkennbar.

```
public void N4_Produktgruppen()
{
 // Berechnung der Produktgruppen für den Fall N = 4

 int N = 4 ;                              // Anzahl der einzelnen Produkte
 int M = 16 - 1 ;                         // Anzahl der Produktmombinationen

 String T[] = {"a","b","c","d"} ;         // Warenkorb mit vier Einzelprodukten
 String R[] = new String[M];              // Vektor der Regeln

 int i = 0;   // Schleifenzähler für die erste  Schleife
 int j = 0;   // Schleifenzähler für die zweite Schleife
 int k = 0;   // Schleifenzähler für die dritte Schleife
 int r = 0;   // Schleifenzähler für die vierte Schleife

 int counter = 0 ;      // Hilfsvariable für die Printausgabe

 for(i=0;i<N  ;i++)   {R[r]=T[i]          ; r=r+1;}    // Einerkombination

 for(i=0  ;i<N-1;i++)
 for(j=i+1;j<N  ;j++) {R[r]=T[i]+T[j]     ; r=r+1;}    // Zweierkombination

 for(i=0  ;i<N-2;i++)
 for(j=i+1;j<N-1;j++)
 for(k=j+1;k<N  ;k++) {R[r]=T[i]+T[j]+T[k]; r=r+1;}    // Dreierkombination

 R[r] = T[0]+T[1]+T[2]+T[3];                           // Viererkombination

 for(r=0;r<M;r++) {counter=r+1;                        // Konsolenausgabe
                   System.out.println(counter+"\t ---> \t"+R[r]);
                  }
}
```

Als Ergebnis erscheint dann die folgende Anzeige auf der Konsole:

```
 1  --->  a
 2  --->  b
 3  --->  c
 4  --->  d
 5  --->  ab
 6  --->  ac
 7  --->  ad
 8  --->  bc
 9  --->  bd
10  --->  cd
11  --->  abc
12  --->  abd
13  --->  acd
14  --->  bcd
15  --->  abcd
```

2.4 Logische Algorithmen zur Warenkorb-Analyse

Die numerische Umsetzung der Warenkorb-Analyse basiert auf einem logischen Warenkorb, der so viel Stellen wie Einzelprodukte N ausweist, wobei jede Stelle das Vorhandensein eines einzelnen Produktes mit einer logischen Eins abbildet. In der Abb. 2.2 enthält der Warenkorb N=4 die Produkte (a,b,d) und das Produkt c fehlt. Der entsprechende logische Warenkorb ergibt sich dann aus einer Reihe von logischen Einsen, wobei an der 3. Stelle für das nicht vorhandene Produkt c eine logische Null auftaucht.

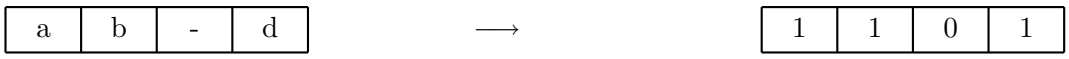


Abb.: 2.2: Logischer Warenkorb

Ein weiterer elementarer numerischer Vereinfachungsschritt besteht darin, ein Hashingverfahren auf die Produktkombinationen anzuwenden. Gemäß der binomischen Formel 2.5 zur Berechnung der Produktgruppen M entstehen $2^N - 1$ Ziffern für den Fall von M Produktkombinationen. Daher ordnet man zu jeder Produktgruppe eine eindeutige Ziffer zu. Z.B. nimmt für den Fall N=4 die Anzahl der Produktkombinationen M den Wert von 15 an, wodurch für einen 4-stelligen Warenkorb einschließlich eines leeren Warenkorbes eindeutig alle 16 Kodierungen möglich sind.

Eine weitere numerisch günstige Randbedingung besteht darin, dass die Reihenfolge der Produktkombinationen in einem Warenkorb ohne Bedeutung ist. Ordnet man einer aufsteigenden Ziffernfolge nun genau jene Produktkombination zu, die seiner logischen Darstellung im binären Zahlensystem entspricht, dann entsteht ein numerisch optimaler Algorithmus, der bei der Analyse von Warenkörben nur noch logische Operatoren ausführt. Die Analyse eines Warenkorbes besteht algorithmisch z.B. für N=4 nur noch in der Feststellung, ob die Ziffern 0 bis 15 in der binären Darstellung des Warenkorbinhaltes vorhanden sind.

Die Anzahl der erforderlichen Schleifen hängt von der maximalen Anzahl der gleichzeitig in einem Warenkorb befindlichen Produkte ab. Betrachtet man ein Sortiment von N Artikeln, dann kauft ein Kunde nie alle Artikel gleichzeitig, sondern wählt maximal K Artikel aus. Unter dieser Bedingung treten erheblich weniger Produktgruppen M auf und es gilt:

$$\text{Anzahl der tatsächlichen Produktgruppen} \quad M = \sum_{k=1}^{K} \binom{N}{k} < 2^N - 1; \quad K < N \tag{2.6}$$

Die erläuterten Techniken erschließen den numerischen Zugang zu sehr schnellen Algorithmen für die Warenkorb-Analyse und ermöglichen realistische Rechenzeit und Speicherplatzanforderungen. Zusammenfassend ging es um die folgenden Techniken:

- Darstellung der Warenkörbe als logische Bitmuster z.B. im Java-Format boolean.
- Jeder Warenkorbinhalt entspricht eindeutig einer Ziffer zwischen Eins und 2^N.
- Für den relevanten Fall $K << N$ reduziert sich der Rechenaufwand erheblich.
- Programmtechnisch erfordert die Warenkorb-Analyse nur noch die Auswertung sehr schneller logischer Vergleichsoperatoren.

Beispiel: Graphische Darstellung der Kodierungen für den Sonderfall N=4

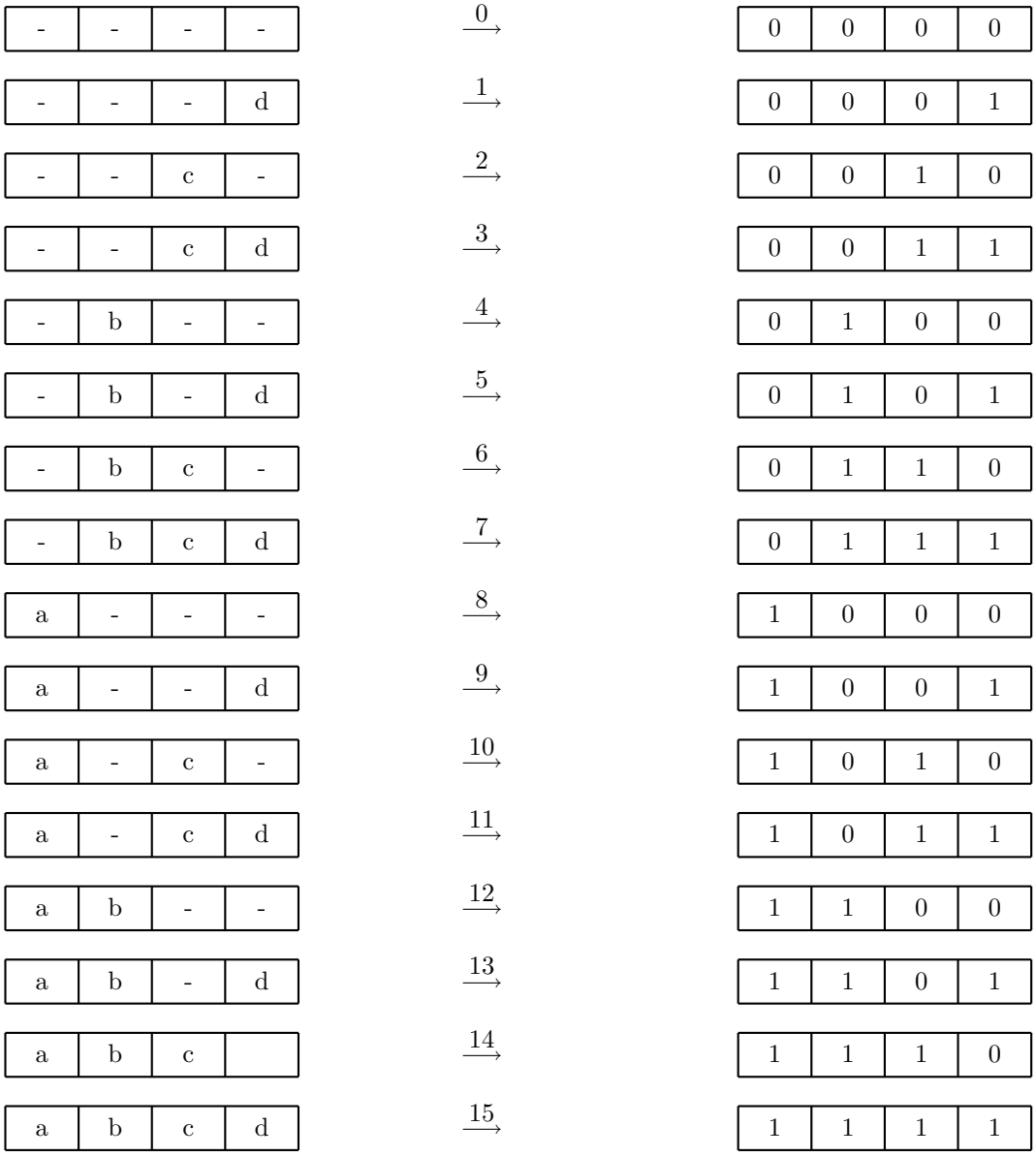


Abb.: 2.3: Hashing eines logischen Warenkorbes für N=4

Beispiel: Analyse eines einzelnen Warenkorbes mit N=4

Der in der Abb. 2.4 dargestellte Warenkorb enthält die Produkte a bzw. d und besteht somit aus zwei Einzelprodukten (a),(d) und der Produktkombination (a,d).

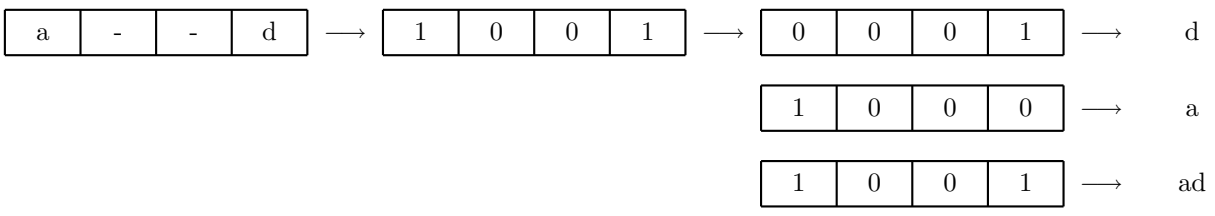


Abb.: 2.4: Auswertung eines einzelnen Warenkorbes für N=4

So wie in der Abb. 2.4 dargestellt, erfolgt für jeden einzelnen Warenkorb die Bestimmung der Produktkombinationen. Ein Häufigkeitsvektor h addiert die jeweilige Anzahl einer Produktkombination und speichert nach Auswertung aller Warenkörbe die absoluten Häufigkeiten. Die Normierung mit der Anzahl der Warenkörbe ergibt den gesuchten relativen Supportvektor s.

Beispiel: Supportberechnung für vier Warenkörbe mit N=4

Die Berechnung des Supportes von vier Warenkörben (Abb. 2.5) gliedert sich in mehrere Stufen:

1. Sortieren der Produkte innerhalb jedes Warenkorbes in alphabetisch aufsteigender Reihenfolge. Falls die Produkte mit Kennziffern bezeichnet sind, ordnet man nach aufsteigenden Kennziffern. Das erste Produkt bzw. das Produkt mit der niedrigsten Kennziffer belegt den ersten Platz im sortierten Warenkorb.

2. Die sortierten Warenkörbe bildet man auf logische Warenkörbe ab, wobei das Vorhandensein eines Produktes einer logische Eins entspricht.

3. Die relativen Häufigkeiten für das Auftreten eines Produktes bzw. einer Produktkombination erscheinen in der Supporttabelle jeweils an der Stelle, die dem Index der logischen Darstellung des Produktes bzw. der Produktkombination entspricht. Im vorliegenden Beispiel gehört zum Produkt (a) der Index 8, d.h. bei dreimaligem Auftreten von (a) ergibt sich der Support von (a) zu $\frac{3}{4}$ bzw. zu 75% und wird an der Stelle 8 aufgetragen.

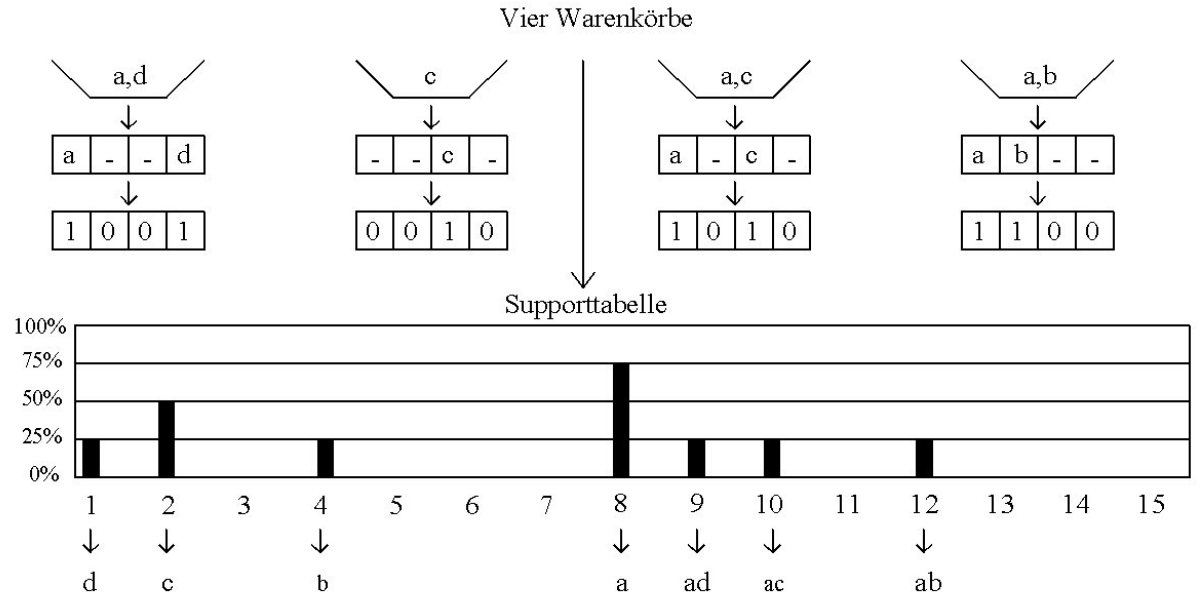


Abb.: 2.5: Beispiel einer Supportberechnung mit logischen Warenkörben

Programmtechnisch erfordert die Supportberechnung für jeden Warenkorb eine Anzahl von M Abfragen, wobei M der Anzahl der praktisch möglichen Produktkombinationen entspricht. Falls die UND-Verknüpfung des logischen Warenkorbes mit einer logischen Produktkombination übereinstimmt, dann enthält der Warenkorb diese Produktkombination und der Zähler für diese Produktkombination erhöht sich.

Der besondere Vorteil dieses Verfahrens zur Bestimmung der Produktkombinationen besteht in der kurzen Rechenzeit, die bei logischen Operationen auf digitalen Rechenanlagen vorliegt. Hinzu kommt der relativ geringe Speicherplatzbedarf, den Formate vom Typ boolean erfordern.

Bei der Programmierung der Algorithmen ist noch zu beachten, dass die Konfidenzberechnungen nur für solche Paarungen von Produktkombinationen definiert sind, die in einem einzelnen Warenkorb auch gemeinsam vorhanden sind. Z.B. entsteht aus dem Produkt (a) und dem Produkt (b) die Produktkombinationen (ab). Daher muss die Summe der einzelnen Produktkombinationen jener Produktkombinationen entsprechen, aus denen sie entstand. Nicht definiert wären z.B. die Produktkombinationen „a“ und „ab“, weil in diesem Fall das Produkt (a) doppelt vorhanden wäre und ein Warenkorb jedes Einzelprodukt nur einmal enthalten darf.

Die Bestimmung der zulässigen Produktkombinationen erfolgt wieder auf der Basis der logischen Warenkörbe. Verbotene doppelte Einzelprodukte treten dann auf, wenn die logischen Einsen der beiden Produktkombinationen an der gleichen Stelle auftauchen. Die logische UND-Verknüpfung der beiden Produktkombinationen erkennt diesen Fall, denn der bitweise UND-Vergleichung darf an keiner Stelle eine logische Eins ergeben.

Beispiel: Verbotene und erlaubte Konfidenz-Produktkombinationen für N=4

Aus der Graphik 2.6 geht hervor, dass sich im mittleren und unteren Fall doppelte Einsen ergeben. Nur der oberste Fall ist zulässig.

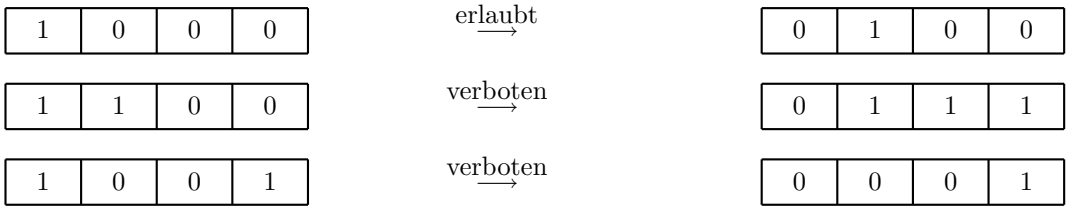


Abb.: 2.6: Verbotene und erlaubte Konfidenz-Produktgruppen für N=4

Die Visualisierung der Support- und Konfidenzwerte erfolgt anschaulich z.B. in Balkendiagrammen oder räumlichen Graphiken. Durch eine entsprechende Farbgebung lassen sich Produktgruppen mit hohen Werten für den Support und für die Konfidenz markieren. Da bereits bei 6 Einzelprodukten insgesamt 602 Paarungen entstehen, stellt man z.B. nur die größten Support und die Konfidenzwerte graphisch dar und vernachlässigt mittels eines einzustellenden Schwellwertes die restlichen Paarungen.

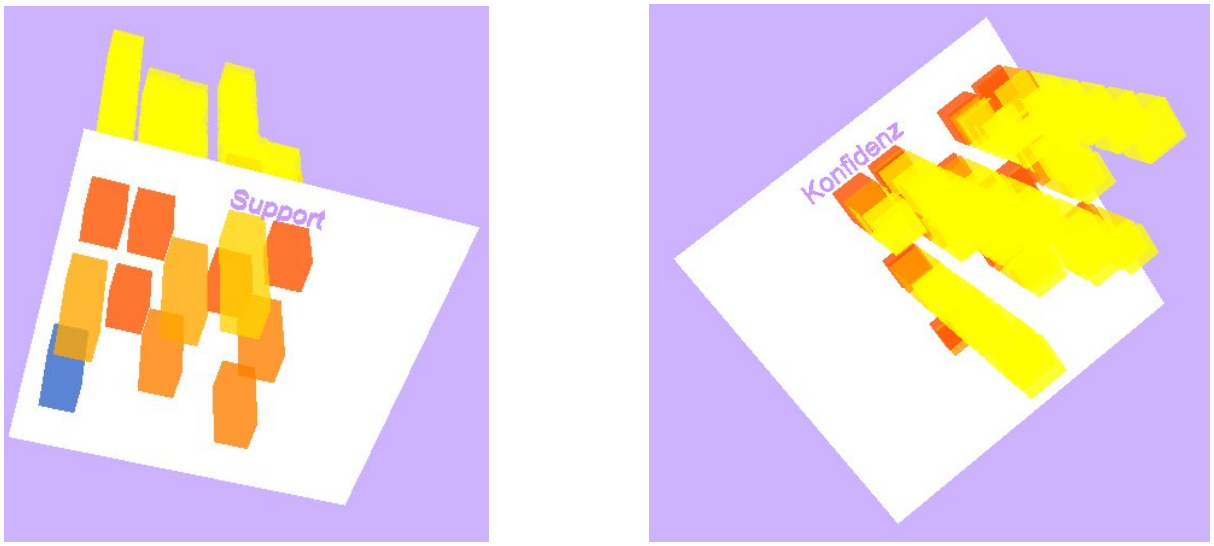


Abb.: 2.7: 3D-Visualisierungstudie zur Darstellung von Support- und Konfidenzwerten

2.5 Die Apriori-Methode mit logischen Warenkörben

Das systematische Weglassen von Regeln, die auf geringen Supportwerten basieren, ist im „Apriori-Algorithmus“ konsequent umgesetzt. Der Algorithmus entstand 1994 in einem IBM Forschungslabor in den USA [16] und erfuhr inzwischen zahlreiche Abwandlungen und Verbesserungen. Die zentrale Idee basiert auf der Tatsache, dass der Support z.B bei einer Produktkombination (abc) immer gleich oder kleiner als die Suppportwerte der einzelnen Untergruppen ist. Im Grenzfall, bei dem z.B. überhaupt kein Einzelprodukt (a) in den Warenkörben mehr vorhanden ist, fallen dann jene Produktgruppen weg, die ein Einzelprodukt (a) enthalten. Das Streichen der Produktgruppen mit zu geringen Supportwerten bezeichnet man als „Pruning“.

Wendet man diesen Zusammenhang auf die Auswertung der Warenkörbe an, dann ist es sinnvoll, zuerst die Supportwerte aller Einzelprodukte zu berechnen. Häufig liegt der eine oder der andere Supportwert unterhalb eines vorzugebenden Grenzwertes und damit fallen alle Produktkombinationen, die ein solches Einzelprodukt enthalten, aus der Auswertung heraus. Im nächsten Schritt bestimmt man die Supportwerte aller noch relevanten Zweier-Produktkombination und wiederholt die Vorgehensweise aus dem ersten Rechenschritt, d.h. falls der Supportwert einer Zweier-Produktkombination einen zu geringen Supportwert aufweist, lassen sich alle Produktkombinationen ab der Dreier-Produktkombination, die diese Zweier-Produktkombination enthalten, wieder vernachlässigen.

Die Rechenzeitersparnis beim Apriori-Algorithmus hängt stark von der Höhe des einzustellenden Grenzwertes für den minimalen Support ab. Je höher dieser Grenzwert, desto schneller läuft der Algorithmus, weil dann besonders viele Produktkombinationen herausfallen. Erfreulich ist, dass bei der Wissensgenerierung besonders die hohen Supportwerte interessieren, denn bei einer ausreichenden Zahl von Warenkörben entspricht der Supportwert der Wahrscheinlichkeit für das Auftreten einer Produktkombination. Daher startet man den Apriori-Algorithmus zunächst mit einem hohen Supportwert und erhält dann relativ schnell die ersten interessanten Regeln. Durch das schrittweise Reduzieren des Supportgrenzwertes entstehen dann bei gleichzeitigem Ansteigen der erforderlichen Rechenzeit ständig weitere Regeln.

Die Bildung der Produktkombinationen erfolgt nur beim Programmstart und belastet daher die online-Auswertung der Warenkörbe nicht mehr. Für das überschaubare Beispiel von vier Einzelprodukten (a,b,c,d) sieht dann der Pool von Produktkombinationen wie in der Tab.: 2.6 dargestellt aus. Die letzte Spalte der Tab.: 2.6 enthält den Pruning-Parameter, dessen Wert für den Fall vernachlässigbarer Produktkombinationen von Eins auf Null wechselt. Für den hier angenommenen Fall, dass nach dem ersten Durchlauf die Supportwerte für die Einzelprodukte (c) und (d) unterhalb des Grenzwertes liegen, geht die Tab.: 2.6 in die Tab.: 2.7 über.

Wie der Blick auf Tab.: 2.7 bestätigt, bleibt für den nächsten Durchlauf, der mit den Zweierkombinationen ab der Nummer 5 beginnt, nur noch die Produktkombination (ab) übrig. Das Komprimieren des Ziffernpools wäre jetzt möglich, allerdings würde dieser Rechenschritt bei der in der Praxis üblichen extrem hohen Anzahl an Ziffern zu einem unvertretbaren Zeitaufwand führen. Daher ist es günstiger, sich an den logischen Pruning-Parameter zu halten und ab der Nummer 5 mit einer if-Abfrage die vernachlässigten Produktkombinationen zu überspringen.

Für das Generieren des Ziffernpools wandelt man die Ziffern zwischen 1 und 2^N in logische Darstellungen um, bestimmt für jede dieser Darstellungen die Anzahl der logischen Einsen und schreibt die logische Darstellung in jene Zeile des Ziffernpools, ab der die Ziffern mit dieser Anzahl gespeichert sind.

Für die Tab.: 2.7 bedeutet dies konkret, dass die Einzelprodukte ab der ersten Zeile, die Zweier-Produktkombinationen ab der 5. Zeile, die Dreier-Produktkombinationen ab der 11. Zeile und die Vierer-Produktkombinationen ab der 15. Zeile auftreten. Eine Speicherung des Ziffernpools in einer Matrixstruktur würde zu extrem unterschiedlichen Zeilenlängen für jede Spalte führen und scheidet daher aus.

laufende Nummer	Produktkombination	logische Darstellung	Ziffer	Pruning-Parameter
1	a	1000	8	1
2	b	0100	4	1
3	c	0010	2	1
4	d	0001	1	1
5	ab	1100	12	1
6	ac	1010	10	1
7	ad	1001	9	1
8	bc	0110	6	1
9	bd	0101	5	1
10	cd	0011	3	1
11	abc	1110	14	1
12	abd	1101	13	1
13	acd	1011	11	1
14	bcd	0111	7	1
15	abcd	1111	15	1

Tab.: 2.6: Ziffernpool vor dem Pruning

Nummer	Produktkombination	log. Darstellung	Ziffer	Pruning-Parameter
1	a	1000	8	1
2	b	0100	4	1
3	c	0010	2	0
4	d	0001	1	0
5	ab	1100	12	1
6	ac	1010	10	0
7	ad	1001	9	0
8	bc	0110	6	0
9	bd	0101	5	0
10	cd	0011	3	0
11	abc	1110	14	0
12	abd	1101	13	0
13	acd	1011	11	0
14	bcd	0111	7	0
15	abcd	1111	15	0

Tab.: 2.7: Ziffernpool nach dem Pruning

Die Berechnung des Offsets für die Speicheradressen basiert auf den binomischen Zahlen, die angeben, wie viele Produktkombinationen, die genau aus k Einzelprodukten bestehen, man aus einer Sammlung von N Einzelprodukten entnehmen kann.

$$\boxed{\text{Offset(K)} = \sum_{k=1}^{K-1} \binom{N}{k} + 1; \quad K \leq N} \tag{2.7}$$

Für das in der Tab.: 2.7 darstellte Beispiel folgt durch das Auswerten der Gl. 2.7 an der Stelle K=4 der offset zu:

$$offset = \sum_{k=1}^{3} \binom{4}{k} + 1 = \frac{4}{1} + \frac{4}{2} = 4 + 6 + 1 = 11$$

Beispiel: Ein Beispiel aus der Literatur

In [2] befindet sich im Kapitel über die Assoziationsmethoden ein anschauliches Beispiel zur Analyse von 9 Warenkörben mit 5 Einzelprodukten I1 bis I5, das auf dem in der Tab.: 2.8 dargestellten Datensatz basiert. Aus der zweiten Spalte ergibt sich die in der 3. Spalten aufgeführte Liste der logischen Warenkörbe. Bei N=5 Einzelprodukten sind insgesamt $2^5 - 1 = 31$ verschiedene Produktkombinationen zu berechnen.

Warenkorbnummer	Warenkorbinhalt	logischer Warenkorb	Datensatzformat
1	I1,I2, I5	11001	1,1,1,0,0,1
2	I2, I4	01010	2,0,1,0,1,0
3	I2, I3	01100	3,0,1,1,0,0
4	I1, I2, I4	11010	4,1,1,0,1,0
5	I1, I3	10100	5,1,0,1,0,0
6	I2, I3	01100	6,0,1,1,0,0
7	I1, I3	10100	7,1,0,1,0,0
8	I1, I2, I3, I5	11101	8,1,1,1,0,1
9	I1, I2, I3	11100	9,1,1,1,0,0

Tab.: 2.8: Warenkorbliste

Im Rahmen des Forschungssemesters wurde ein Java-Programm entwickelt, welches das Apriori-Konzept auf der Basis logischer Warenkörbe und logischer Pruning-Operationen implementiert. Das Format der Eingangsdaten beginnt mit der laufenden Nummer des Warenkorbes und listet dann, jeweils durch ein Komma getrennt, den Inhalt der Warenkörbe in logischer Darstellung auf. Die Tab.: 2.8 enthält in der letzten Spalte den auf diese Weise formatierten Datensatz.

Für einen Grenzwert von 10% für den minimalen Support stellt die Abb. 2.8 die Ergebnisse der Auswertung des Datensatzes (Tab.: 2.8) dar.

Die Ergebnisse für die Konfidenzwerte hängen von den Grenzwerten des Supports ab, weil die Division von zwei kleinen Supportwerten zu hohen Konfidenzwerten führen kann. Deshalb sollte man den Grenzwert für den Support relativ groß wählen. Die Tab.: 2.9 stellt die nach der Größe sortierten Konfidenzwerte für einen erhöhten Support-Grenzwert von 20% dar.

Wie aus der Tab.: 2.8 ersichtlich, besitzt das Produkt I2 mit 78% den höchsten Supportwert. Daher schlägt man dann in der Tab.: 2.9 die dazu gehörenden Konfidenzwerte nach und gelangt somit zu folgender Aussage:

Wer das Produkt I2 erwirbt, interessiert sich mit einer Wahrscheinlichkeit von 56% auch für die Produkte I1 oder I3.

0	4	(I2)	[01000]	7	78%
1	3	(I3)	[00100]	6	67%
2	5	(I1)	[10000]	6	67%
3	11	(I2,I3)	[01100]	4	44%
4	14	(I1,I3)	[10100]	4	44%
5	15	(I1,I2)	[11000]	4	44%
6	1	(I5)	[00001]	2	22%
7	2	(I4)	[00010]	2	22%
8	9	(I2,I5)	[01001]	2	22%
9	10	(I2,I4)	[01010]	2	22%
10	12	(I1,I5)	[10001]	2	22%
11	23	(I1,I2,I5)	[11001]	2	22%
12	25	(I1,I2,I3)	[11100]	2	22%
13	7	(I3,I5)	[00101]	1	11%
14	13	(I1,I4)	[10010]	1	11%
15	18	(I2,I3,I5)	[01101]	1	11%
16	21	(I1,I3,I5)	[10101]	1	11%
17	24	(I1,I2,I4)	[11010]	1	11%
18	29	(I1,I2,I3,I5)	[11101]	1	11%

Abb.: 2.8: Warenkorb-Analyse für einen Support-Grenzwert von 10%

Nummer	Prämisse	Konklusion	Konfidenzwert
0	(I5)	(I2)	100%
1	(I5)	(I1)	100%
2	(I5)	(I1,I2)	100%
3	(I4)	(I2)	100%
4	(I2,I5)	(I1)	100%
5	(I1,I5)	(I2)	100%
6	(I3)	(I2)	66%
7	(I3)	(I1)	66%
8	(I1)	(I2)	66%
9	(I1)	(I3)	66%
10	(I2)	(I3)	56%
11	(I2)	(I1)	56%
12	(I2,I3)	(I1)	50%
13	(I3)	(I1,I2)	33%
14	(I1)	(I2,I3)	33%
15	(I1)	(I5)	33%
16	(I1)	(I2,I5)	33%
17	(I2)	(I1,I3)	28%
18	(I2)	(I5)	28%
19	(I2)	(I4)	28%
20	(I2)	(I1,I5)	28%

Tab.: 2.9: Konfidenzwerte bei 10% minimalem Support

2.6 Benchmark für die Apriori-Methode

Die allgemeinen Algorithmen zur Warenkorb-Analyse führten zu sehr langen Rechenzeiten, die z.B. bei N=15 Einzelprodukten und 10000 Warenkörben bei ca. einer Stunde lagen. Das Pruning der kleinen Supportwerte ermöglicht dagegen erheblich schnellere Algorithmen. Am Beispiel von simulierten Datensätzen, deren Inhalt ein Zufallsgenerator bestimmt, lassen sich daher Benchmark-Rechenläufe starten und der Rechenzeitbedarf protokollieren. Die Ergebnisse sind in der Tab.: 2.10 dargestellt.

Test Nr.	Einzel-produkte N	Anzahl der Warenkörbe	Produkt-kombinationen	Produkte K je Warenkorb	Support-Grenzwert	Rechenzeit h:m:s
1	15	10000	32.768	15	10%	0: 0:43
2	20	10000	1.048.576	20	10%	0:41:19
3	20	10000	1.048.576	10	10%	0: 1:39
4	20	10000	1.048.576	10	50%	0: 0:16
5	20	10000	1.048.576	20	20%	0: 8:02
6	20	20000	1.048.576	20	10%	0:41:47

Tab.: 2.10: Benchmark-Rechenläufe

Die Testläufe bestätigen die extreme Abhängigkeit der benötigten Rechenzeit von den drei wesentlichen Parametern einer Warenkorb-Analyse:

- Anzahl N der Einzelprodukte
- maximale Anzahl K der Produkte eines Warenkorbes
- minimaler Grenzwert für den Support

Die Anzahl N der Einzelprodukte definiert über den Zusammenhang 2^N die Anzahl der Produktkombinationen, deren Häufigkeit für jede einzelne Produktkombination zu prüfen ist. Während beim Test 1 nur ca. 32.000 Varianten auftreten, steigt diese Zahl bei den Testläufen 2 bis 8 auf ca. 1.000.000 an und erreicht beim Testlauf 8 für N=23 schon den Wert von ca. 8.000.000. Die Testläufe 1 und 2 unterscheiden sich nur durch eine Erhöhung von N=15 auf N=20, wodurch jedoch die Rechenzeit von knapp einer Minute auf ca. 41min ansteigt.

Die Anzahl K beschreibt die maximale Anzahl der Produkte, die in einem Warenkorb gleichzeitig vorkommen. Da ein Kunde bei vielen Anwendungen erheblich weniger Produkte, als vorhanden sind, erwirbt, reduziert sich damit die Zahl der möglichen Produktkombinationen, was sich wiederum günstig auf die Rechenzeit auswirkt. Die Rechenzeit des Testlaufes 3 sinkt von ca. 41min auf ca. 2min für den Testlauf 3 ab, wenn man den Parameter K=20 auf K=10 reduziert.

Ein hoher minimaler Support wirkt sich günstig auf die Rechenzeit aus, denn das Pruningverfahren arbeitet dann besonders effektiv. Der Testlauf 4 unterscheidet sich nur durch den Wert von 50% für den minimalem Support gegenüber dem Testlauf 3, bei dem dieser Wert bei 10% lag. Die Rechenzeit sinkt dadurch von ca. 100s auf nur 16s ab. Gegenüber dem Testlauf 2, der eine Rechenzeit von ca. 42min verbrauchte, bedeutet die Verdopplung des minimalen Supportwertes von 10% auf 20% eine Verringerung der Rechenzeit von ca. 42min auf ca. 8min, wie sich für den Testlauf 5 ablesen lässt.

Der Testlauf 6 entspricht dem Testlauf 2, wobei nun die doppelte Anzahl von Warenkörben vorliegt. Die erforderliche Rechenzeit steigt dadurch nur um ca. eine halbe Minute an, was gegenüber der gesamten Rechenzeit von ca. 42min nicht ins Gewicht fällt.

Zusammenfassend bleibt festzuhalten, dass man die Rechenläufe stets mit einem möglichst großen Schwellwert für den Support beginnen sollte, denn für diesen Fall erhält man sofort Konfidenzwerte, die eine erste Aussage ermöglichen. Dies stellt im eigentlichen Sinne keine Näherung dar, weil die Regeln, die auf großen Supportwerten basieren, den Anwender auch am meisten interessieren.

Programmtechnisch gesehen, geht man zu Beginn der Rechenläufe von einem Grenzwert für den Support von 100% aus und senkt dann diesen Wert, z.B. in Stufen von 10%, in einer Schleife so lange ab, bis die Konfidenzwerte ebenfalls einen Schwellwert von z.B. 25% überschreiten. Damit erhält man bei der Warenkorb-Analyse die wichtigste Aussage in der kürzesten Rechenzeit und kann dem Kunden noch während seines Bestellvorganges die entsprechenden Hinweise geben.

Weitere Rechenläufe mit geringeren Schwellwerten für den Support dienen zur Optimierung der Geschäftsprozesse, für den kein Echtzeitkriterium gilt.

Kapitel 3

Klassifizierung

Der Begriff der Klassifizierung kennzeichnet jene Methoden, welche einen aktuellen Datensatz einer bestimmten Klasse zuordnen. Die Mitglieder dieser Klasse besitzen alle eine wichtige gemeinsame, technisch oder betriebswirtschaftliche Eigenschaft, wie z.B. eine definierte Qualitätsstufe oder eine geplante Rendite.

Die Klassen mit ihrem Attributen sind in Form von Trainingsdaten gegeben und liegen häufig in einer Tabellenform, wie z.B. einem Excel-Arbeitsblatt, vor. Ein typisches Beispiel stellt die Klasse jener Bankkunden dar, die ein bestimmtes Finanzprodukt erwerben und die sich von einer anderen Kundenklasse, welche nur Informationsmaterial angefordert hat, vor allem durch das „Zielattribut“ dieser Klassen, nämlich Kauf bzw. Nichtkauf, unterscheiden. Alle Kunden mit dem gleichen Zielattribut bilden gemeinsam eine bestimmte Klasse. Die Methoden der Klassifizierung liefern eine Wahrscheinlichkeit für die Zuordnung eines neuen Kunden, dessen Klasse nicht bekannt ist, zu der entsprechenden Klasse.

Die Methoden der Klassifizierung bestehen aus vier Schritten:

1. Auswahl von informativen Rohdaten
2. Aufbereitung der Rohdaten
3. Entwicklung der entsprechenden Algorithmen
4. Berechnung, zu welcher Klasse ein neuer Datensatz gehört (Klassifizierung)

Der eigentlichen Klassifizierung, welche näherungsweise in Echtzeitbedingungen erfolgen sollte, geht die Auswahl und die Aufbereitung der Rohdaten sowie die Entwicklung der entsprechenden Algorithmen voraus, was einen erheblichen Rechenaufwand bedeuten kann, der jedoch nicht unter Echtzeitanforderungen steht. Die wichtigsten Verfahren des Klassifizierens sind:

- Entscheidungsbaumverfahren
- Regressionsverfahren
- Support Vektor Verfahren
- Neuronale Klassifizierungsnetze

In den letzten Jahren traten vor allem die Support Vektor Verfahren und die künstlichen neuronalen Netze immer mehr in den Vordergrund.

3.1 Auswahl von informativen Rohdaten

Häufig stehen Rohdaten zur Verfügung, die zunächst eine viel zu große Anzahl von unterschiedlichsten Attributen aufweisen. Die Anzahl der Attribute bestimmt jedoch den für die Klassifizierung erforderlichen Rechenaufwand. Zu vermeiden sind in jedem Fall jene Attribute, die sich nur wenig auf die zu bestimmenden Klassen auswirken. Die Tab.: 3.1 stellt beispielhaft einen Ausschnitt (10 Datensätze) aus einem Rohdatenbestand dar.

Nr.	Einkommen	Altersgruppe	Sternzeichen	Wohnort		Häufg krank(Klasse)
1	30000	30	Widder	Hannover		ja
2	30000	30	Krebs	Berlin		nein
3	30000	20	Fisch	Freiburg		ja
4	30000	30	Widder	Hannover		nein
5	30000	20	Krebs	Celle		ja
6	30000	20	Fisch	Freiburg		nein
7	30000	40	Widder	Berlin		ja
8	30000	40	Fisch	Freiburg		nein
9	30000	20	Krebs	Celle		ja
10	30000	40	Fisch	Hannover		ja

Tab.: 3.1: Auswirkung von Attributen auf das Klassen-Attribut „Krankheitstage“

Offensichtlich spielt in diesem Beispiel das Attribut „Einkommen“ keine Rolle bei der Klassifizierung. Allerdings gilt dies nur, weil weitere Datensätze mit unterschiedlichen Werten für dieses Attribut fehlen. Zusätzliche Datensätze mit variablen Einkommenswerten könnten die erforderlichen Informationen beisteuern.

Doch wie sieht es mit dem Attribut „Sternzeichen“ aus ? Hier helfen mathematische Verfahren weiter, die eine quantitative Aussage über die Auswirkung eines Attributs auf die Klasse ermöglichen. Besonders geeignet dazu sind die beiden folgenden Methoden:

- Berechnung des Informationsgewinns der Attribute
- Berechnung des Korrelationskoeffizienten der Attribute

Bei der Auswahl geeigneter Datensätze bzw. Attribute kommt es darauf an, jene Daten auszuwählen, die einen hohen Informationsgewinn ergeben und eine starke Korrelation zwischen den Attributen und der Klasse aufweisen. Die Berechnung der Korrelationskoeffizienten liefert eine quantitative Aussage über den Zusammenhang eines Attributes mit der Klasse. Allerdings muß dazu ein kausaler Zusammenhang zwischen beiden bestehen. So könnte sich durchaus für das Attribut des Sternzeichens ein von Null verschiedener Korrelationskoeffizent ergeben, der dann allerdings durch Plausibilitätsbetrachtungen verworfen werden sollte.

Die Attribute gliedern sich nach innen in Stufen. Für die nachfolgenden Berechnungen spielt die relative Häufigkeit p, mit der eine bestimmte Stufe innerhalb eines Attributs auftritt, eine große Rolle. So erscheint z.B. in der Tab.: 3.1 beim Attribut „Wohnort“ die nominale Stufe „Hannover“ zweimal , d.h. für die vorgegebenen sechs Datensätze ergibt sich daraus eine relative Häufigkeit von $p = \frac{2}{6}$. Für das Data Mining interessant sind Datensätze mit möglichst hohen Werten von p. Je mehr Stufen ein Attribut aufweist, umso aufwändiger gestaltet sich die Entwicklung der Algorithmen und die Rechenzeit- und Speicherplatzanforderungen nehmen erheblich zu.

3.1.1 Informationsgewinn eines Attributs

Der Informationsgewinn G(A) eines Attributes A lässt sich gemäß der Informationstheorie über den Logarithmus zur Basis 2 mittels den Gl. 3.1 in der Einheit Bits über die Entropie D wie folgt berechnen:

a	= Zähler für die Attribute
k	= Zähler für die Stufen
$p_K(k)$	= relative Häufigkeit der Stufe k in der Klasse K
$p_A(k,K)$	= relative Häufigkeit der Stufe k mit der Klasse K im Attribut A
$NDaten$	= Anzahl aller Datensätze
$NStufen(a)$	= Anzahl der Stufen, die im Attribut vorhanden sind
$KStufen$	= Anzahl der Stufen in der Klassenspalte
$NAttribut$	= Anzahl der Attribute
$AnzahlK(k)$	= Anzahl der Datensätze der Klassenspalte für die Stufe k
$AnzahlA(k,K)$	= Anzahl der Datensätze der Stufe k mit der Klasse K im Attribut A
$AnzahlN(k,a)$	= Anzahl aller Datensätze der Stufe k im Attribut a

$$
\begin{aligned}
p_K(k) &= \frac{AnzahlK(k)}{NDaten} \\
D &= -\sum_{k=1}^{KStufen} [p_K(k) \cdot \log_2(p_K(k))] \\
p_A(k,K) &= \frac{AnzahlA(k,K)}{AnzahlN(k,a)} \\
G(a) &= D + \sum_{k=1}^{NStufen(a)} \left[\frac{AnzahlN(k,a)}{NDaten} \sum_{K=1}^{KStufen} [p_A(k,K) \log_2 (p_A(k,K))] \right]
\end{aligned}
\tag{3.1}
$$

Beispiel: Berechnung des Informationsgewinnes

Die in der Tab.: 3.1 dargestellten Datensätze ermöglichen die anschauliche Berechnung des Informationsgewinns. Ein Blick auf die rechte Spalte, welche die Klassen mit den beiden Stufen „ja“ und „nein“ enthält, zeigt, dass von den insgesamt 10 Datensätzen 6-mal die Stufe „ja“ und viermal die Stufe „nein“ erscheint. Damit folgt dann für die Entropie:

$$
\begin{aligned}
p_K(Stufe = ja) &= \tfrac{6}{10} \quad \text{bzw.} \quad p_K(Stufe = nein) = \tfrac{4}{10} \\
D &= -p_K(ja) \cdot \log_2(p_K(ja)) - p_K(nein) \cdot \log_2(p_K(nein)) \\
D &= -\tfrac{6}{10} \log_2(\tfrac{6}{10}) - \tfrac{4}{10} \log_2(\tfrac{4}{10}) = 0.970951 \approx 97\%
\end{aligned}
$$

Würde nur eine einzige Stufe vorhanden sein, dann ergäbe sich $p_K = 1$ und somit nähme wegen log(1)=0 die Entropie den Wert von Null an. Gesucht sind jedoch solche Datensätze, deren Klassen-Entropie möglichst groß ist. Für den anderen Sonderfall gleichverteilter Klassen, d.h. $p_K(ja) = p_K(nein) = \frac{1}{2}$ ergibt sich der maximale Wert der Entropie zu 100%.

$$
D = -2 \left(\frac{1}{2} \log_2(\frac{1}{2}) \right) = -\log_2(\frac{1}{2}) = \log_2(2) = 1
$$

Der Informationsgewinn ist für jedes Attribut eigens zu berechnen. Wählt man z.B. das dreistufige Altersattribut aus und sortiert die Datensätze nach den Stufen in diesem Attribut, dann folgt die Tab.: 3.2 und alle Parameter für die Berechnung des Informationsgewinnes liegen fest.

Altersgruppe	Anzahl der Datensätze	Anzahl von ja-Klassen	Anzahl von nein-Klassen
20	4	3	1
30	3	1	2
40	3	2	1

Tab.: 3.2: Sortieren der Datensätze nach dem Attribut „Altersgruppe“

$$
\begin{aligned}
NDaten &= 10 \\
p_A(Altersgruppe = 20, ja) &= \frac{\text{AnzahlA(Klasse,ja)}}{\text{AnzahlN(Altersgruppe=20,Alter)}} = \tfrac{3}{4} \\
p_A(Altersgruppe = 20, nein) &= \frac{\text{AnzahlA(Klasse,nein)}}{\text{AnzahlN(Altersgruppe=20,Alter)}} = \tfrac{1}{4} \\
p_A(Altersgruppe = 30, ja) &= \frac{\text{AnzahlA(Klasse,ja)}}{\text{AnzahlN(Altersgruppe=30,Alter)}} = \tfrac{1}{3} \\
p_A(Altersgruppe = 30, nein) &= \frac{\text{AnzahlA(Klasse,nein)}}{\text{AnzahlN(Altersgruppe=30,Alter)}} = \tfrac{2}{3} \\
p_A(Altersgruppe = 40, ja) &= \frac{\text{AnzahlA(Klasse,ja)}}{\text{AnzahlN(Altersgruppe=40,Alter)}} = \tfrac{2}{3} \\
p_A(Altersgruppe = 40, nein) &= \frac{\text{AnzahlA(Klasse,nein)}}{\text{AnzahlN(Altersgruppe=40,Alter)}} = \tfrac{1}{3}
\end{aligned}
$$

$$
\begin{aligned}
G(A) = D \quad & + \tfrac{4}{10}\left(\tfrac{3}{4}\log_2(\tfrac{3}{4}) + \tfrac{1}{4}\log_2(\tfrac{1}{4})\right) \\
& + \tfrac{3}{10}\left(\tfrac{1}{3}\log_2(\tfrac{1}{3}) + \tfrac{2}{3}\log_2(\tfrac{2}{3})\right) \\
& + \tfrac{3}{10}\left(\tfrac{2}{3}\log_2(\tfrac{2}{3}) + \tfrac{1}{3}\log_2(\tfrac{1}{3})\right) \\
& = 0.0945 \approx 9\%
\end{aligned}
\tag{3.2}
$$

Auf diese Weise sind für alle Attribute der jeweilige Informationsgewinn zu berechnen. Für die Tab.: 3.1 stellt die Tab.: 3.3 die restlichen Ergebnisse in Prozent dar.

Einkommen	Altersgruppe	Sternzeichen	Wohnort
0%	9%	2%	22%

Tab.: 3.3: Informationsgewinn der Attribute

Die Tab.: 3.3 weist den größten Informationsgewinn von 22% beim Attribut „Wohnort“ aus. Das Sternzeichen wirkt sich in vernachlässigbarer Weise auf die Krankheitstage aus. Das Attribut „Einkommen“ besitzt, wie erwartet, keinerlei Informationsgewinn.

Die relativen Häufigkeiten p konvergieren mit einer zunehmenden Zahl von Datensätzen gegen die Wahrscheinlichkeiten. Schon aus diesem Grund kommt es bei der Auswahl von Datensätzen und Attributen darauf an, möglichst große Häufigkeiten für das Auftreten einzelner Stufen in den verschiedenen Attributen zu erhalten.

Bei analogen Zahlenwerten erfolgt die Diskretisierung in n Stufen, wobei ein zu geringes n einen Informationsverlust verursacht und ein großer Wert von n zu kleinen Werten bei den relativen Häufigkeiten führt. Man kann sich in diesem Zusammenhang an der Abschätzungsformel aus dem Gebiet der mathematischen Statistik orientieren, die für N Datensätze lautet:

$$n \approx 2\sqrt[3]{N} \tag{3.3}$$

Für einen Datenbestand mit z.B. 1000 Datensätzen empfiehlt die Abschätzung gemäß der Gl. 3.3 ungefähr 20 Stufen. Falls ein Attribut des Datenbestandes zu viele Stufen besitzt, fasst man direkt aufeinander folgende Stufen zu einer gemeinsamen Stufe zusammen. Umgekehrt sollte man bei zu wenig Stufen die Datenerhebung differenzierter gestalten und einfach mehr Stufen erfassen.

Der bisher berechnete Informationsgewinn hängt von der Anzahl der Stufen ab, was sich bei Attributen mit einer unterschiedlichen Anzahl von Stufen als nachteilig beim Vergleich der Informationsgewinne der Attribute auswirkt. Daher empfiehlt es sich, den Informationsgewinn jedes Attributes mit dem Informationsgewinn der Attributstufen zu normieren. Der Normierungsfaktor DNorm ergibt sich gemäß der Gl. 3.4 zu:

$$p_{Stufe}(k,a) = \frac{AnzahlN(k,a)}{NDaten}$$

$$DNorm(a) = -\sum_{k=1}^{NStufen(a)} [p_{Stufe}(k,a) \cdot \log_2(p_{Stufe}(k,a))] \tag{3.4}$$

$$G_{relativ}(a) = \frac{G(a)}{DNorm(a)}$$

Ein Blick in die Tab.: 3.1 liefert die Zahlenwerte zur beispielhaften Berechnung der Normierung für das Attribut „Alter“, das aus 3 Stufen besteht und 4, 3 und 3 entsprechende Datensätze je Altersstufen aufweist.

$$DNorm(\text{„Alter“}) = -\frac{4}{10}\log_2(\frac{4}{10}) - \frac{3}{10}\log_2(\frac{3}{10}) - \frac{3}{10}\log_2(\frac{3}{10}) = 1.5709 \tag{3.5}$$

Mit dieser Normierung reduziert sich für das Attribut „Alter“ der Informationsgewinn, für den die Gl.: 3.2 einen Wert von ca. 9% ergab, auf den kleineren Wert von ca. 6%.

$$G_{relativ}(\text{„Alter“}) = \frac{0.0945}{1.5709} \approx 6\% \tag{3.6}$$

Bei konkreten Datenbeständen mit vielen Attributen trägt jedes Attribut nur mit einem relativ geringen Informationgewinn zur vorliegenden Klasse bei. Informationsgewinne über 20% sind eher selten, jedoch kann ein unerwarteter hoher Informationsgewinn für ein bestimmtes Attribut auf einen sehr interessanten Zusammenhang hinweisen und unter Umständen eine „Goldader“ im Datenbestand darstellen. Das Unternehmen „Weight-Watchers“ stieß auf eine solche „Goldader“ durch den berechneten überraschend hohen Informationsgewinn des Attributes „Haustiere vorhanden“ bei der Data Mining Analyse von Personendaten zur Gewinnung von Neukunden [36].

3.1.2 Korrelationskoeffizient eines Attributes

Der Begriff der Korrelation stammt aus dem Gebiet der mathematischen Statistik und beschreibt den Grad des inneren Zusammenhanges zwischen zwei Messreihen. Für Klassifizierungsverfahren entsprechen diese beiden Messreihen den Datensätzen für ein bestimmtes Attribut und den Datensätzen der Klasse. Für das Beispiel der Datensätze, die in der Tab.: 3.1 aufgelistet sind, folgen z.B. für das Attribut „Alter“ die beiden in der Tab.: 3.4 enthaltenen Messreihen, wobei die Klassenstufe ja dem Wert 1 und die Klassenstufe nein dem Wert 2 entspricht.

Messreihe 1	30	30	20	30	30	20	40	40	20	40
Messreihe 2	1	2	1	2	1	2	1	2	1	1

Tab.: 3.4: Messreihen zur Korrelationsberechnung für das Attribut „Alter“

Die Berechnung des Korrelationskoeffizienten erfordert zunächst die Bestimmung der Kreuzkovarianz. Dazu multipliziert man paarweise die einzelnen Stufen der beiden Messreihen und summiert die Produkte auf. Die Normierung der Kreuzkovarianz mit den jeweiligen beiden Autovarianzen ergibt schließlich den Wert des Korrelationskoeffizienten. Mathematisch beschreibt diesen Algorithmus in Form eines Gleichungssatzes die Gl. 3.7:

$$
\begin{aligned}
&\text{1. Messreihe mit n Werten} \quad X = \{x_1, x_2, \ldots, x_n\} \\
&\text{2. Messreihe mit n Werten} \quad Y = \{y_1, y_2, \ldots, y_n\} \\
&\bar{x} = \tfrac{1}{n} \sum_{i=1}^{n} x_i \qquad\qquad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \\
&\sigma_{xx}^2 = \tfrac{1}{n} \sum_{i=1}^{n} (x_i \cdot x_i) - (\bar{x})^2 \qquad \sigma_{yy}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i \cdot y_i) - (\bar{y})^2 \\
&\sigma_{xy}^2 = \tfrac{1}{n} \sum_{i=1}^{n} (x_i \cdot y_i) - \bar{x} \cdot \bar{y} \\
&\text{Korrelationskoeffizient} \rho = \frac{\sigma_{xy}^2}{\sqrt{\sigma_{xx}^2 \cdot \sigma_{yy}^2}} \cdot 100\% \quad \text{mit} \quad -100\% \leq \rho \leq 100\%
\end{aligned}
\tag{3.7}
$$

Die Tab.: 3.5 enthält Korrelationskoeffizienten, die sich aus der Anwendung der Gl. 3.7 auf die Datensätze der Tab.: 3.1 ergeben und stellt zum Vergleich in der ersten Zeile die entsprechenden Informationsgewinne (Tab.: 3.3) den Korrelationskoeffizienten gegenüber.

	Einkommen	Altersgruppe	Sternzeichen	Wohnort
Informationsgewinn	0%	9%	2%	22%
Korrelationskoeffizient	0%	10%	15%	-11%

Tab.: 3.5: Vergleich Informationsgewinn und Korrelationskoeffizient der Attribute

Selbstverständlich nimmt der Wert des Korrelationskoeffizienten für das konstante Attribut „Einkommen“ ebenfalls den Wert 0 an. Das negative Vorzeichen beim Attribut „Wohnort“ weist auf einen umgekehrt proportionalen Zusammenhang des Wohnortes zur Klasse hin und ist in diesem Zusammenhang ohne Bedeutung, denn die nominalen Wohnorte wurden für die numerische Rechnung frei wählbaren Kennzahlen zugeordnet. Das Korrelationsverfahren bewertet das Attribut „Sternzeichen“ höher als der Informationsgewinn dies aussagt. Allerdings setzt die Interpretation eines Korrelationskoeffizienten immer eine große Anzahl von Datensätzen voraus, was in diesem einfachen, dafür aber anschaulichen Beispiel, mit nur 10 Datensätzen nicht gegeben ist.

3.2 Aufbereitung der Rohdaten

Rohdatensatze stammen aus bestimmten Sichten einer Datenbank und liegen häufig in unterschiedlichen Datenformaten vor. Fehlerhafte oder fehlende Einzelattribute und mehrfach vorhandene, identische Datensätze (Doubletten) sind dabei eher die Regel als die Ausnahme. Ein Ausschnitt eines Trainingsdatensatzes könnte z.B., so wie in der Tab.: 3.6 dargestellt, aussehen.

Nr.	Einkommen	Alter	Steuerklasse	Wohnort	weitere Attribute		Zielattribut
1	50000	411	1	Hannover	...		true
2	20000	54	?	München	...		true
3	500000	24	3	Bremen	...		false
⋮	⋮	⋮	⋮	⋮	⋮		⋮
n	10000	66	1	Celle	...		true

Tab.: 3.6: Fehlerhafte Trainingsdaten mit fehlenden Attributen

Die Tab.: 3.6 verdeutlicht beispielhaft die Probleme, die bei der Auswertung von Trainingsdatensatzen zu lösen sind:

- In den Trainingsmengen fehlen einzelne Datenelemente
  (Datensatz 2, Angabe der Steuerklasse fehlt)
- Die Trainingsdaten enthalten Tippfehler, fehlerhafte Angaben und unsinnige Werte
  (Datensatz 1, Lebensalter unsinnig)
- Alle der Formate sind auf Zahlenwerte abzubilden und sinnvoll zu skalieren
  (Die Angaben 500000, 1, Hannover und true müssen z.B. auf einen Zahlenbereich, z.B. zwischen -100 und 100, skaliert werden.)

Die Aufbereitung der Rohdaten erfordert zunächst eine Plausibilitätsprüfung, welche neben dem eigentlichen Definitionsbereich der Attribute auch auf extreme Abweichungen der Attribute von statistischen Kriterien, wie Mittelwert und Varinanz, achtet. Als hilfreich haben sich graphische „Preview“-Darstellungen der einzelnen Attribute erwiesen, weil beim Betrachten der entsprechenden Verläufe fehlerhafte Datensätze optisch auffallen.

Fehlende Attribute lassen sich aus jenem Datensatz übernehmen, der zu dem lückenhaften die größte Ähnlichkeit aufweist oder durch Mittelwerte ersetzen. Bei einer ausreichenden Anzahl von Datensätzen bietet sich immer ein Überspringen eines als fehlerhaft erkannten Datensatzes an.

Bei Attributen, die sich numerisch wenig verändern, lässt sich der Mittelwert eliminieren und statt dem absoluten Wert nur die relative Änderung des Attributs auf seinen Maximalwert beziehen.

Logische Attribute, wie sie meistens für die beiden Stufe der Klasse üblich sind, interpretiert man z.B. als -100 bzw. 100. Ähnlich geht man bei Bewertungsangaben wie z.B., Handelsklasse I, II, III oder VI, vor und ordnet dem Attribut jeweils eine ganze Zahl zu.

Bei Textangaben bietet sich eine Indizierung an. So könnte man in der Tab.: 3.6 die Ortsnamen durch einen ganzzahligen Wert austauschen und somit die Städte einfach fortlaufend nummerieren.

Bei den skalierten Datensätzen vermeidet möglichst Nullwerte, denn die nachfolgende algorithmische Verarbeitung liefert bei einer Multiplikationsoperation ebenfalls den Wert Null und damit ist die Information dieses Attributes verloren.

Die Auswahl der Skalierungsmethoden hängt letztlich von der Art der vorliegenden Rohdaten ab. Grundsätzlich geht es bei der Aufbereitung der Rohdaten darum, folgende Forderungen zu erfüllen:

- Numerisches Skalieren der Rohdaten auf einen einheitlichen Definitionsbereich der Attribute
- Erhalt der vollständigen Information der Rohdaten
- Ausschluss von falschen Informationen, welche durch fehlerhafte Rohdatensätze entstehen können
- Ersatz von numerisch ungünstigen Formaten in den Rohdaten, wie zu große oder zu kleine Zahlenbereiche für die Attribute
- Weitgehende Vermeidung von Nullen für die Attributswerte

Beispiel: Aufbereitung eines Rohdatensatzes

Die Tab.: 3.7 stellt als Beispiel einen Datensatz aus dem Wettbewerb des Data Mining Cups 2006 dar. Insgesamt bestand der Rohdatensatz aus 26 Attributen. 12 Attribute entfielen wegen ihrer geringen Auswirkung auf die Klasse. Die Prozentwerte basieren auf den jeweiligen Maximalwerten der Attribute von 8000 Datensätzen. Beim Artikel dieser EBay-Auktion handelt es sich um MP3-Player, die in den Attributen mit den Nummern 1 bis 6 in Form von Textbeschreibungen spezifiziert sind.

Nummer	Bedeutung	Rohdatenwert	Skalierter Wert
7	Dauer der Auktion	10	10%
9	Rating des Verkäufers	2	20%
10	Startpreis	1	1%
11	Sofortkaufpreis	100	100%
13	mit Fettschrift	ja	100%
15	mit Kategorie-Überschrift	ja	100%
16	mit Bilddarstellung	nein	0%
18	mit Animation	nein	0%
20	mit Hintergrundfarbe	ja	100%
21	mit Startzeitpunkt	ja	100%
23	Menge des Artikels	50	20%
24	erzielter Erlös	75	75%
25	durchschnittlicher Erlös	90	90%
26	Verkauf mit Gewinn	nein	0%

Tab.: 3.7: Beispiel eins skalierten Datensatzes

Eine sorgfältige Aufbereitung der Rohdaten ermöglicht erst den Erfolg der nachfolgenden eigentlichen Klassifizierung. Fehlerhafte Datensätze oder ungünstig skalierte Attributsstufen, die sich kaum auf die Klasse auswirken, verursachen falsche Klassifizierungen und zwar unabhängig vom jeweils gewählten Algorithmus. Deshalb ist eine aufwändige Aufbereitung der Rohdaten unbedingt erforderlich.

3.3 Entscheidungsbäume

Einen zum Klassifizieren geeigneten Entscheidungsbaum entwickelt man in Richtung des größten Informationsgewinnes oder alternativ auch in Richtung des größten Korrelationskoeffizienten. Beginnend mit dem Rootknoten 0 liefern die Formeln 3.4 bzw. 3.7 für jedes Attribut einen Wert. „Gewonnen“ hat das Attribut mit dem maximalen Wert und aus seinen m Stufen entstehen dann m weitere Knoten 1,2,...,m. Diese m Knoten liegen in der Tiefe=1 des entstehenden Entscheidungsbaumes.

Am Knoten 1 steht nun ein in reduzierter Datensatz zur Verfügung, der die Bedingung der ersten Attributstufe des Gewinnerattributs aus dem Knoten 0 enthält. Entsprechend gehören zum Knoten 2 die Datensätze mit der zweiten Attributsstufe. Dem Knoten m sind die Datensätze mit der Attributsstufe m des Gewinnerattributs zugeordnet.

Die weitere Entwicklung des Entscheidungsbaumes wiederholt sich sinngemäß für die nächste Tiefe=2. Wieder ist der Gewinner zu ermitteln und entsprechend den Attributsstufen des Gewinnerattributes die nächste Tiefe des Entscheidungsbaumes zu bestimmen. Bei jeder weiteren Tiefe fällt das Gewinnerattribut des „Vaterknotens“ weg und am Ende bestehen die Knoten in der letzten Tiefe nur noch aus einem einzigen Attribut und der Klasse.

Der Wert der Klasse in der letzten Tiefe stellt die gesuchte Lösung beim Klassifizieren eines aktuellen Datensatzes dar. Dazu parst man den gespeicherten Graphen des Entscheidungsbaumes von oben nach unten und gibt den Wert der Klasse am Ende des Entscheidungsbaumes aus.

In der Regel enden viele Entwicklungen nicht in der letzten Tiefe, sondern, wegen mangelnder Datensätze oder wegen nicht vorhandener Informationsgewinne bzw. fehlender Korrelationen in den Datensätzen, in geringeren Tiefen. Das Ende eines „Zweiges“ im Entscheidungsbaum nennt in Analogie zu einem natürlichen Baum ein „Blatt“. .

Offensichtlich handelt es sich hier um einen einfachen Algorithmus, der jedoch erheblichen Programmieraufwand erfordert und bei vielen Attributen mit zahlreichen Stufen jeden Speicher zum Überlaufen bringt. Der Algorithmus eignet sich daher eher für einfach strukturierte Datensätze. Als vorteilhaft erweist sich jedoch, dass die aufwändige Entscheidungsbaum-Entwicklung nur beim Programmstart oder bei einer geänderten Datenbasis erfolgt.

Das eigentliche Problem beim Entwickeln eines Entscheidungsbaumes besteht darin, eine optimale Skalierung der Datensätze zu finden, denn die Anzahl der Stufen beeinflusst direkt die Gestalt des Entscheidungsbaumes. Viele Stufen erfordern eine entsprechende große Anzahl von Datensätzen, die häufig gar nicht vorhanden sind. Wenige Stufen generieren eine eher einfache Gestalt des Entscheidungsbaumes, der jedoch wegen der dann grob strukturierten Informationen zu einer geringen Klassifizierungsrate führt.

Viele Attribute erhöhen die Tiefe des Entscheidungsbaumes, denn die maximale Tiefe entspricht exakt der Anzahl der Attribute. Dies kann bei einer begrenzten Anzahl von Datensätzen zu Blättern führen, die bereits in geringeren Tiefen auftreten, was bedeutet, dass bei der Auswertung des Entscheidungsbaumes nicht mehr alle vorhandenen Attribute abgefragt werden.

Aus den eben genannten Gründen lassen sich Entscheidungsbäume immer dann effektiv einsetzen, wenn die beiden folgenden Bedingungen zutreffen:

- Der Datenbestand umfasst mehr als 10000 Datensätze
- Die Anzahl der Attribute ist auf ca. 10 relevante Attribute begrenzt

3.3.1 Beispiel einer Entscheidungsbaum-Entwicklung

Die Tab.: 3.8 stellt als Beispiel einen Datensatz dar, der aus Gründen der Anschaulichkeit nur aus 10 Datensätzen besteht.

Nr.	Alter	Sternbild	Wohnort	krank
1	7	1	5	10
2	6	5	4	10
3	1	3	3	1
4	4	4	6	10
5	8	2	7	10
6	2	3	3	1
7	1	1	2	1
8	3	2	3	1
9	3	3	4	1
10	3	4	4	10

Tab.: 3.8: Beispiel mit 10 Datensätzen

Aus diesem Datensatz folgt eine Datenstufen-Matrix (Tab.: 3.9), welche für jedes Attribut die Stufen angibt. Die letzte Zeile der Datenstufen-Matrix enthält die Anzahl der Stufen zur Entwicklung des Entscheidungsbaumes. Die beiden einzigen Stufen der Klasse ergeben daher eine 2 in der letzten Spalte und Zeile. Das Attribut „Alter“ weist mit 7 Datenstufen den höchsten Wert auf. Ausgehend von der Tab.: 3.9 ermittelt man die insgesamt 3

Nr.	Alter	Sternbild	Wohnort	krank
1	7	1	5	10
2	6	5	4	1
3	1	3	3	0
4	4	4	6	0
5	8	2	7	0
6	2	0	2	0
7	3	0	0	0
8	7	5	6	2

Tab.: 3.9: Datenstufen-Matrix für 10 Datensätze

Korrelationskoeffizienten gemäß dem Gleichungssatz 3.7 und erhält die folgenden, in der Tab.: 3.10 dargestellten Ergebnisse: Mit 78% nimmt das Attribut „Alter“ gegenüber den

	Alter	Sternbild	Wohnort
Korrelationskoeffizient	78%	32%	76%

Tab.: 3.10: Korrelationskoeffizienten am Rootknoten für 10 Datensätze

anderen beiden Attributen den größten Wert an und ausgehend von diesem Vaterknoten 0 entstehen nun 7 neue Kindknoten, denn das Attribut „Alter“ gliedert sich in 7 Stufen. Ein Blick auf die erste Spalte mit dem Attribut „Alter“ in der Tab.: 3.8 zeigt, dass nur die Stufen 1 und 3 mehr als einen Datensatz enthalten. Die Stufen 2,4,5,6 und 7 führen deshalb zu Knoten, die als Blatt im Entscheidungsbaum keine weitere Entwicklung mehr zulassen.

Die Stufe 1 entspricht ebenfalls einem Blatt, denn die entsprechenden Werte der Klasse sind beim Datensatz 3 und 7 in beiden Fällen 1 und somit identisch, was für alle verbliebenen Attribute einen Korrelationskoeffizienten von Null ergibt.

Die Entwicklung des Entscheidungsbaumes setzt sich nun am Knoten 7 mit der Alterstufe=3 fort, für den insgesamt 3 Datensätze (8,9,10) mit unterschiedlichen Klassen (1,1,10) vorliegen. Ausgangspunkt für die Korrelationsberechnungen am Knoten 7 stellen wieder die Gleichungssätze 3.7 dar. Als Ergebnis der Korrelationsrechnung folgt ein Wert von 87% für das Attribut „Sternzeichen“, der gegenüber dem Wert des Attributs „Wohnort“ von 50% als Gewinner auftritt.

Nr.	Sternzeichen	Wohnort	krank
8	2	3	1
9	3	4	1
10	4	4	10

Tab.: 3.11: Datensätze am Knoten 7 für die Stufe=3 des Attributs „Alter“

Die Entwicklung des Entscheidungsbaumes geht nun mit den Stufen des Attributs „Sternzeichen“ weiter. Allerdings liegt nun für jede der 5 Stufen des Attributs „Sternzeichen“ nur noch ein einziger Datensatz vor und damit endet der Entscheidungsbaum in den Knoten 10,11 und 12, die als Blätter gelten.

Da in diesem Beispiel nur 10 Datensätze vorliegen, würde z.B. ein zu klassifizierender Datensatz, der bis zum Knoten 7 mit dem Datensatz der Tab.: 3.11 gelangt, bei einem Sternzeichen=2 und einem Wohnort=3 als Klasse=1 erkannt. Falls jedoch für einen neuen zu klassifizierenden Datensatz z.B. die Stufe Sternbild=1 erreicht würde, fehlt der erforderliche Datensatz bei der Entwicklung des Entscheidungsbaumes und eine Klassifizierung dieses Datensatzes ist nicht mehr möglich. Damit bestätigt sich auf anschauliche Weise, dass konkrete Anwendungen aus einer Vielzahl von „informativen“ Datensätzen bestehen und nicht zu viele Stufen aufweisen sollten.

Den kompletten Entscheidungsbaum für dieses Beispiel mit 10 Datensätzen zeigt die Abb.: 3.1. Die Abkürzungen S, W und K stehen für Sternbild, Wohnort und Klasse. Die jeweilige Knotennummer ist an der linken obere Ecke jedes Knotens vermerkt. Die Kreise kennzeichnen ein Blatt des Entscheidungsbaumes.

Das Klassifizieren beginnt stets mit dem Knoten 0. Die Ziffer 1 in der ersten Spalte weist auf das Attribut „Alter“ hin, das mit 78% den größten Korrelationskoeffizienten erreicht hatte. Je nach der vorliegenden Altersstufe, geht es dann mit dem entsprechenden Knoten weiter. Liegt z.B. bei dem zu klassifizierenden Datensatz eine Stufe Alter=3 vor, dann setzt sich das Parsen mit dem Knoten 3 fort. Am Knoten 3 gibt es keinen Gewinner, denn der Korrelationskoeffizient nimmt in beiden Fällen den Wert 0 an. Damit entspricht die Klasse dieses Datensatzes dem in der ersten Spalte der Graphen-Matrix abgelegten Wert von 1.

	Alter	Sternbild	Wohnort
Normierter Informationsgewinn	27%	27%	31%

Tab.: 3.12: Informationsgewinn am Rootknoten für 10 Datensätze

Der in der Abb.: 3.1 dargestellte Entscheidungsbaum lässt auch anschaulich erkennen, dass bei anderen Werten für den Informationsgewinn bereits ab dem Knoten 0 ein ganz anderer Baum entsteht.

Entwickelt man den Entscheidungsbaum auf der Basis des Informationsgewinnes, dann liefert die Normierung schon beim Rootknoten eine andere Aufteilung, die in der Tab.: 3.12 durch den Vergleich mit den Werten in der Tab.: 3.10 zu erkennen ist. In der Fachliteratur ([1],[2]) sind weitere unterschiedliche Maßzahlen für den Informationsgehaltes eines Attributes beschrieben.

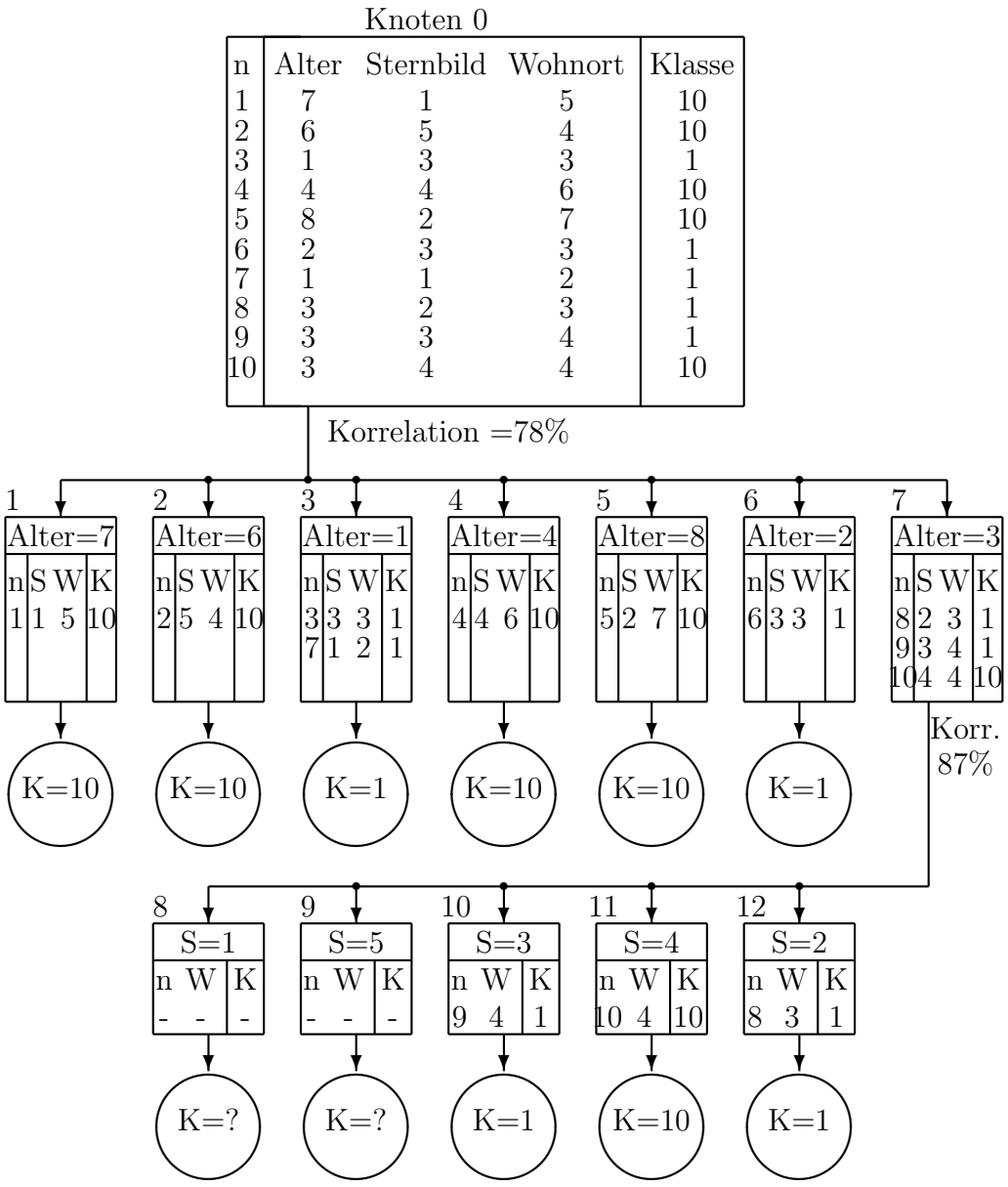


Abb.: 3.1: Entscheidungsbaum (K=Klasse, S=Sternbild, W=Wohnort)

3.3.2 Die Graphen-Matrix

Die Ergebnisse der Entwicklung des Entscheidungsbaum speichert man in einer Graphen-Matrix. Für das Beispiel mit 10 Datensätzen sieht diese Matrix, wie in der Tab.: 3.13 dargestellt, aus. Jede Zeile beschreibt die Situation an einen Knoten mit der Zeilennummer als Knotennummer. Der Knoten 0 entwickelt die Knoten 1 bis 7 und weist somit KZähler=7 auf.

Die Spalte „KZähler“ speichert den aktuellen Stand des Knotenzählers bei der Entwicklung des Entscheidungsbaumes. In der Spalte „VKtn.“ legt man die Nummer des jeweiligen Vaterknotens ab, wobei der Knoten 0 in dieser Spalte alternativ die Gesamtzahl der Knoten des Entscheidungsbaumes speichert.

Den maximalen Wert des Korrelationskoeffizienten kann man in der äußerst rechten Spalte ablesen. Der Gewinner des Startknotens 0 erreichte einen Korrelationskoeffizienten von 78%.

Die erste Spalte der Graphen-Matrix enthält im Fall eines Blattes die Klasse, die dem zugeordneten Datensatz entspricht. Falls kein Blatt vorliegt, gibt der Wert der ersten Spalte die Nummer des Gewinnerattributs an diesem Knoten an. Die weiteren Spalten enthalten für die Stufen des Gewinnerattributs die Knotennummern für die nachgeordneten Kindknoten. Diese Angaben steuern das Parsen bei der Klassifizierung neuer Datensätze.

Einen längeren Durchlauf durch den Entscheidungsbaum löst ein zu klassifizierender Datensatz aus, der eine Stufe Alter=7 enthält. Der Algorithmus springt dabei vom Knoten 0 zum Knoten 7 und nun kommt es auf die Stufe des Attributes „Sternbild“ an. Der Knoten 7 konnte mit einem Korrelationskoeffizienten von 87% für 5 Stufen in die Knoten 8 bis 12 weiter entwickelt werden. Dann ist aber auch hier Schluss, denn alle diese Knoten enden als Blatt. Die Stufen 3,4 und 2 führen über die Knoten 10, 11 und 12 auf die Klassen 1, 10 oder 1.

Für die Stufen 1 und 5 im Attribut „Sternbild“ ist wegen der geringen Anzahl der vorhandenen Datensätze eine Klassifizierung nicht mehr möglich.

Nr.	Kl. Att.	Kn.	Kn.	Kn.	Kn.	Kn.	Kn.	Kn.	Kinder-zahl	Knoten-zähler	Vater-knoten	Korr.-Koeff.
0	1	1	2	3	4	5	6	7	7	7	(12)	78/A
1	10	0	0	0	0	0	0	0	0	7	0	0
2	10	0	0	0	0	0	0	0	0	7	0	0
3	1	0	0	0	0	0	0	0	0	7	0	0
4	10	0	0	0	0	0	0	0	0	7	0	0
5	10	0	0	0	0	0	0	0	0	7	0	0
6	1	0	0	0	0	0	0	0	0	7	0	0
7	0	8	9	10	11	12	0	0	5	12	0	87/S
8	0	0	0	0	0	0	0	0	0	12	7	0
9	0	0	0	0	0	0	0	0	0	12	7	0
10	1	0	0	0	0	0	0	0	0	12	7	0
11	10	0	0	0	0	0	0	0	0	12	7	0
12	1	0	0	0	0	0	0	0	0	12	7	0

Tab.: 3.13: Graphen-Matrix für 10 Datensätze

3.3.3 Graphische Darstellung eines Entscheidungsbaumes

Die Abb.: 3.2 zeigt die Bildschirmdarstellung des Data Mining Open Source Tools „RapidMiner“ für einen Entscheidungsbaum, der auf einem Datenbestand von 14 Datensätzen, 4 Attributen und 2 Klassen basiert. Es handelt sich dabei um das Beispiel „Golfen“ aus der Demosammlung des „RapidMiners“. Als Attribute treten meteorologische Daten auf. Die Klassifizierung unterscheidet die beiden Klassen: Golfspielen oder nicht Golfspielen.

golf.data (1)	golf.data (2)	golf.data (3)	golf.data (4)	golf.data (5)
Outlook	Temperature	Humidity	Wind	Play
attribute	attribute	attribute	attribute	label
nominal	integer	integer	nominal	nominal
single_value	single_value	single_value	single_value	single_value
sunny	85.0	85.0	false	no
sunny	80.0	90.0	true	no
overcast	83.0	78.0	false	yes
rain	70.0	96.0	false	yes
rain	68.0	80.0	false	yes
rain	65.0	70.0	true	no
overcast	64.0	65.0	true	yes
sunny	72.0	95.0	false	no
sunny	69.0	70.0	false	yes
rain	75.0	80.0	false	yes
sunny	75.0	70.0	true	yes
overcast	72.0	90.0	true	yes
overcast	81.0	75.0	false	yes
rain	71.0	80.0	true	no

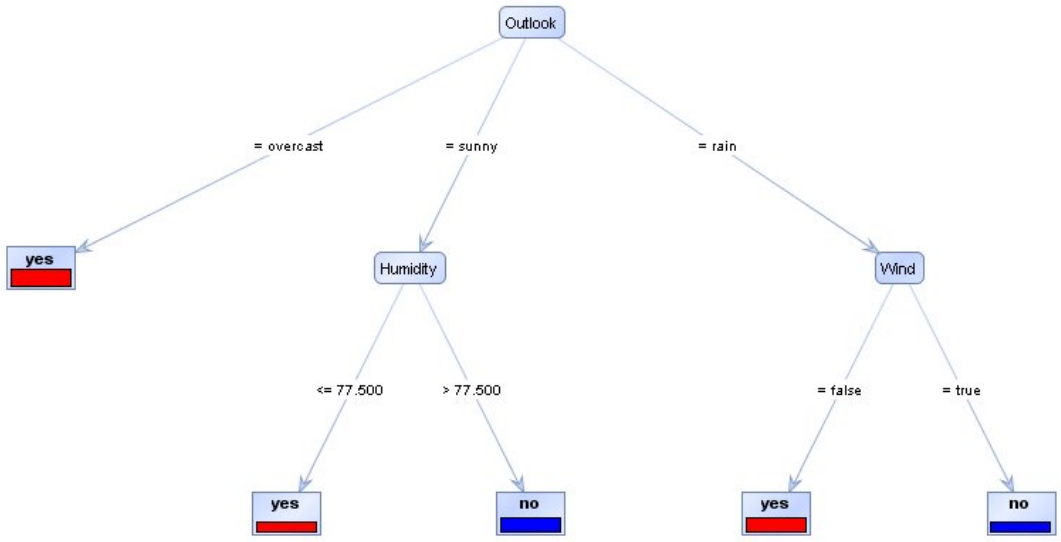


Abb.: 3.2: Beispiel eines Entscheidungsbaumes

3.4 Klassifizieren mit Regressionsverfahren

Regressionsverfahren gehen auf die minimalen Fehlerquadrate des berühmten Mathematikers Gauss zurück. Der Regressionsalgorithmus erfordert, ebenso wie alle anderen Klassifizierungsverfahren, Datensätze, die aus mehreren Attributen und einer Spalte für die Klasse bestehen. Der dreistufige Regressionsalgorithmus basiert auf den Teilaufgaben:

1. Aufteilung der N Datensätze in M Teile entsprechend der Anzahl der Klassen. Enthalten die gegebenen Datensätze z.B. zwei Stufen (true,false), dann entstehen zwei neue Dateien, wobei die eine Datei alle Datensätze mit der Klasse=true enthält und die andere Datei alle Datensätze mit der Klasse=false aufnimmt. Die Anzahl der Datensätze je Teildatenbestand beträgt dann n_m mit $m = 1, \ldots, M$, wobei die Summe über alle n_m der Anzahl aller Datensätze N entspricht.

2. Jede der M Dateien wird durch ein geeignetes mathematisches Modell approximiert, das im einfachsten Fall einer Geraden entspricht, normalerweise jedoch ein nichtlineares mehrdimensionales Gleichungssystem erfordert. Die Anzahl der zu bestimmenden Parameter einer einzelnen Gleichung stimmt mit der Anzahl der Anzahl der Attribute überein.

3. Bei der eigentlichen Klassifizierung bestimmt man die M Abstände des neuen Datensatzes zu den M mathematischen Modellen, wobei der minimale Abstand der erkannten Klasse entspricht.

Der Erfolg des Verfahrens hängt von der Eignung der mathematischen Modelle ab. Nimmt man z.B. Datensätze mit 3 Attributen (x,y,z) an, dann entspricht das mathematische Modell anschaulich einer Fläche im 3D-Raum. Für eine Ebene mit den zunächst unbekannten Parametern (a_1, a_2, a_3) und N Datensätzen lässt sich z.B. das Attribut z als Funktion der restlichen Attribute x und y für das Modell mit m=1 als linearer Zusammenhang wie folgt annehmen:

$$a_1 \cdot x_i + a_2 \cdot y_i + a_3 = z_i; \quad \text{mit} \quad i = 1, ..., n_m \tag{3.8}$$

Das in der Gl. 3.8 definierte Gleichungssystem besteht aus n_m Gleichungen für die Unbekannten a_1, a_2, a_3. Wählt man für das nächste mathematische Modell m=2 eine räumliche Parabel aus, dann entsteht das Gleichungssystem 3.9 zur Berechnung der Koeffizienten b.

$$b_1 \cdot x_i + b_2 \cdot y_i + b_3 \cdot x_i^2 + b_4 \cdot y_i^2 + b_5 = z_i; \quad \text{mit} \quad i = n, ..., n_m \tag{3.9}$$

Für jeden der M Teildatenbestände ist ein anderes mathematisches Modell zu wählen.

Zur Klassifizierung eines einzelnen Datensatzes k (x_k, y_k, z_k) setzt man dessen Attributswerte in alle M mathematischen Modelle ein und bestimmt den Fehler, der proportional zum Abstand eines Datensatzes vom Modell ist.

$$\begin{aligned}
&\text{Abstand } d_1 = \parallel a_1 \cdot x_k + a_2 \cdot y_k + a_3 - z_k \parallel \\
&\text{Abstand } d_2 = \parallel b_1 \cdot x_k + b_2 \cdot y_k + b_3 \cdot x_k^2 + b_4 \cdot y_k^2 + b_5 - z_k \parallel \\
&\text{Abstände } \vdots = \parallel \vdots \parallel \\
&\text{Abstand } d_M = \parallel \cdots \parallel
\end{aligned} \tag{3.10}$$

Die gesuchte Klasse stimmt mit dem Index des minimalen Abstandes überein. Der minimale Abstand d_{min} ermöglicht in Relation zum größten Abstand d_{max} eine quantitative Aussage über die Qualität r der Klassifizierung.

$$r = \left(1 - \frac{d_{min}}{d_{max}}\right) \cdot 100\% \quad \text{mit} \quad 0 \ \leq \ r \ \leq 100\% \tag{3.11}$$

3.4.1 Räumliche Trennung von Datensätzen

Die Abb. 3.3 stellt simulierte ideale Datensätze mit drei Attributen und zwei Klassen dar. Als mathematisches Modell dienen in diesem Fall ein linearer Ansatz. Die obere und untere Begrenzungsfläche trennt die Datensätze, die in der Graphik als Punkte erscheinen. Die mittlere Fläche eignet sich als Kriterium zur Klassifizierung neuer Datensätze, deren Klasse der Lage oberhalb bzw. unterhalb dieser Fläche entspricht.

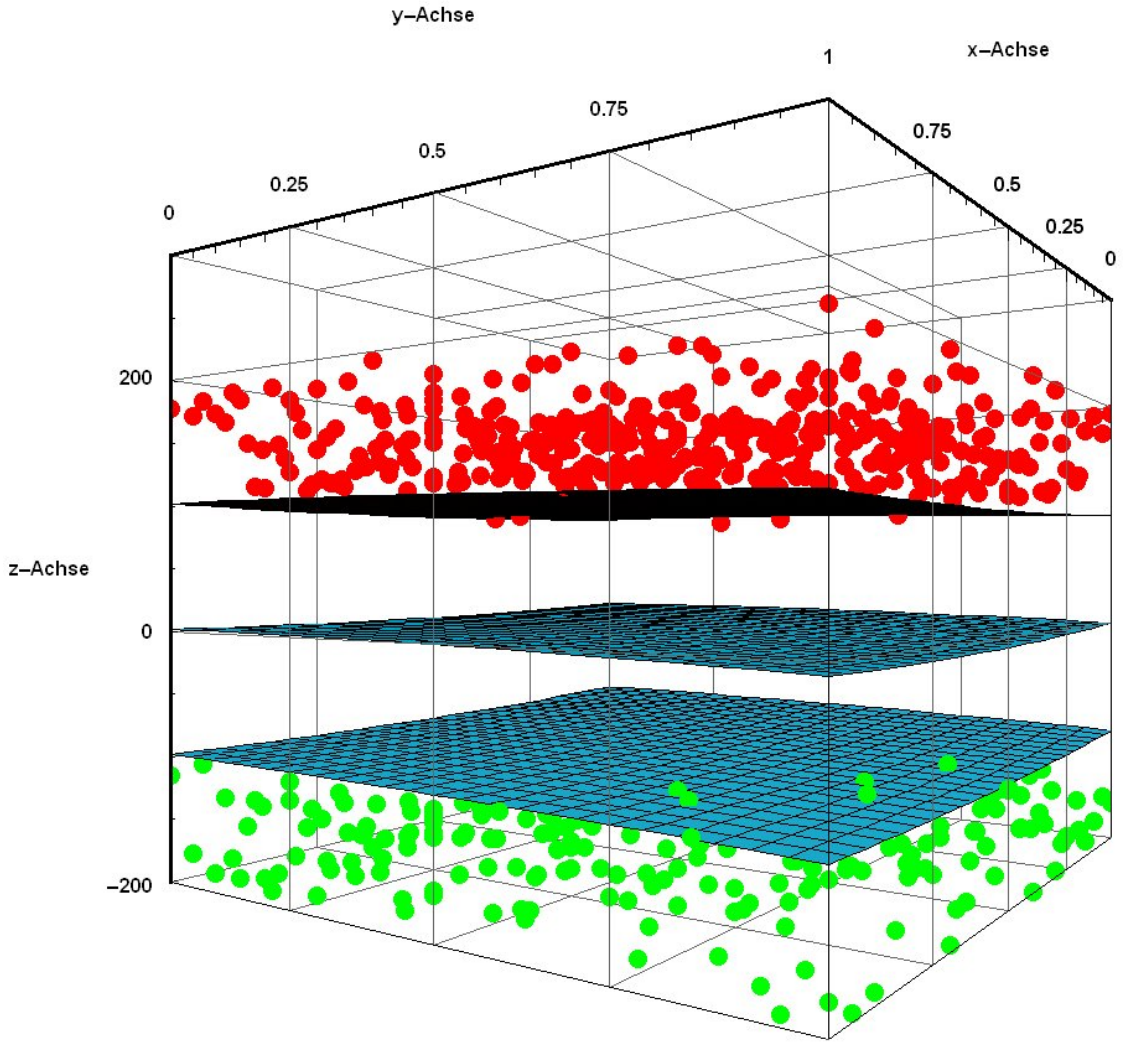


Abb.: 3.3: Trennung von Punktwolken durch einen Regressionsalgorithmus

Die roten Punkte visualisieren Datensätze der Klasse=1 und die grünen Punkte weisen die Klasse=2 auf. In diesem idealen Beispiel gelingt daher die Klassifizierung eines neuen Punktes in korrekter Weise.

Bei praktischen Aufgaben der Klassifizierung liegen leider keine idealen Bedingungen vor. Hier schneiden sich die Punktwolken mehrfach und es kommt darauf an, durch eine geschickte Wahl geeigneter mathematischer Modelle am Ende gut trennbare Punktwolken zu erzeugen.

Die bisherigen Ausführungen zum Regressionsalgorithmus lassen sich auf höhere Dimensionen leicht erweitern. Die Flächen bezeichnet man dann als Hyperflächen. Sie weisen gegenüber dem n-dimensionalen Raum eine um eine Ordnung reduzierte Dimension auf.

3.4.2 Beispiel einer Regressionsanalyse

Die Tab.: 3.14 stellt eine Datei dar, die aus Gründen der Anschaulichkeit nur aus 10 Datensätzen besteht und bereits im Abschnitt über die Klassifikation mittels Entscheidungsbaum als Beispiel diente. Der Vorteil dieser Datei besteht darin, dass sie nur 3 Attribute enthält und sich deshalb die einzelnen Datensätze als Punkte im 3D-Raum anschaulich visualisieren lassen.

Nr.	Alter	Sternbild	Wohnort	Klasse(krank)
1	7	1	5	10
2	6	5	4	10
3	1	3	3	1
4	4	4	6	10
5	8	2	7	10
6	2	3	3	1
7	1	1	2	1
8	3	2	3	1
9	3	3	4	1
10	3	4	4	10

Tab.: 3.14: Beispiel mit 10 Datensätzen

Die Zerlegung dieser Datensätze in zwei Gruppen mit identischen Klassen ergibt die folgenden beiden Dateien, die aus jeweils 5 Datensätzen bestehen:

Nr.	Alter	Sternbild	Wohnort
1	7	1	5
2	6	5	4
4	4	4	6
5	8	2	7
10	3	4	4

Tab.: 3.15: Datensätze mit der Klasse=10

Nr.	Alter	Sternbild	Wohnort
3	1	3	3
6	2	3	3
7	1	1	2
8	3	2	3
9	3	3	4

Tab.: 3.16: Datensätze mit der Klasse=1

Nimmt man für die beiden erforderlichen Hyperflächen jeweils eine Parabelfläche an und interpretiert das Attribut „Wohnort“ als z-Komponente, dann bietet dieses einfache Beispiel den interessanten Sonderfall direkt invertierbarer Matrizen, denn die Zahl der Unbekannten stimmt mit der Zahl der Gleichungen überein. Für den Sonderfall dieser Datensätze liegen alle Punkte exakt auf den beiden jeweiligen Hyperflächen.

Für die beiden linearen Gleichungssysteme Gl. 3.12 bzw. Gl. 3.13 und die zu berechnenden Koeffizienten der beiden Hyperflächen gilt:

$$\begin{pmatrix} 7 & 49 & 1 & 1 & 1 \\ 6 & 36 & 5 & 25 & 1 \\ 4 & 16 & 4 & 16 & 1 \\ 8 & 64 & 2 & 4 & 1 \\ 3 & 9 & 4 & 16 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 6 \\ 7 \\ 4 \end{pmatrix} \tag{3.12}$$

$$\begin{pmatrix} 1 & 1 & 3 & 9 & 1 \\ 2 & 4 & 3 & 9 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 3 & 9 & 2 & 4 & 1 \\ 3 & 9 & 3 & 9 & 1 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 2 \\ 2 \\ 4 \end{pmatrix} \tag{3.13}$$

Ein Mathematiktool liefert die Lösungsvektoren dieser linearen Gleichungssysteme.

$$a = \begin{pmatrix} \frac{159}{34} \\ -\frac{13}{34} \\ \frac{219}{34} \\ -\frac{115}{102} \\ -\frac{730}{51} \end{pmatrix} \quad \text{bzw.} \quad b = \begin{pmatrix} -\frac{3}{2} \\ \frac{1}{2} \\ -\frac{3}{2} \\ \frac{1}{2} \\ 4 \end{pmatrix} \tag{3.14}$$

Für einen neuen zu klassifizierenden Datensatz k setzt man dessen Attributswerte in beide Hyperflächen ein.

$$\begin{aligned} &\text{neuer Datensatz} c = (x_k, y_k, z_k) \\ &\text{Fehler 1} \ = \parallel \tfrac{159}{34} \cdot x_k - \tfrac{13}{34} \cdot x_k^2 + \tfrac{219}{34} \cdot y_k - \tfrac{115}{102} \cdot y_k^2 - \tfrac{730}{51} - z_k \parallel \\ &\text{Fehler 2} \ = \parallel -\tfrac{3}{2} \cdot x_k + \tfrac{1}{2} \cdot x_k^2 - \tfrac{3}{2} \cdot y_k + \tfrac{1}{2} \cdot y_k^2 + 4 - z_k \parallel \end{aligned} \tag{3.15}$$

Da jede Hyperfläche einer bestimmten Klasse zugeordnet ist, weist der geringere Fehler diesem Datensatz die erkannte Klasse zu. Für den Wert r gilt dann:

$$\begin{aligned} &f_{min} = min(\text{Fehler 1}, \text{Fehler 2}) \\ &f_{max} = max(\text{Fehler 1}, \text{Fehler 2}) \\ &r = \left(1 - \tfrac{f_{min}}{f_{max}}\right) \cdot 100\% \quad \text{mit} \quad 0 \ \leq \ r \ \leq 100\% \end{aligned}$$

Wählt man als Test z.B. den ersten Datensatz (7,1,5) aus den Trainingsdaten aus und führt dafür die Klassifizierung durch, dann ergibt sich - wie erwartet - ein Wert von r=100% für die erste Klasse.

$$\begin{aligned} f_1 \ &= \parallel \tfrac{159}{34} \cdot 7 - \tfrac{13}{34} \cdot 7^2 + \tfrac{219}{34} \cdot 1 - \tfrac{115}{102} \cdot 1^2 - \tfrac{730}{51} - 5 \parallel = 0 \\ f_2 \ &= \parallel -\tfrac{3}{2} \cdot 7 + \tfrac{1}{2} \cdot 7^2 - \tfrac{3}{2} \cdot 1 + \tfrac{1}{2} \cdot 1^2 + 4 - 5 \parallel = 12 \\ r_1 &= \left(1 - \tfrac{0}{12}\right) \cdot 100\% = 100\% \quad \text{bzw.} \quad r_2 = \left(1 - \tfrac{12}{12}\right) \cdot 100\% = 0\% \end{aligned}$$

Die Abb.: 3.4 zeigt die beiden Hyperflächen zunächst einzeln in getrennten Darstellungen. Rote Punkte markieren die Datensätze der ersten Klasse und grüne die der zweite Klasse.

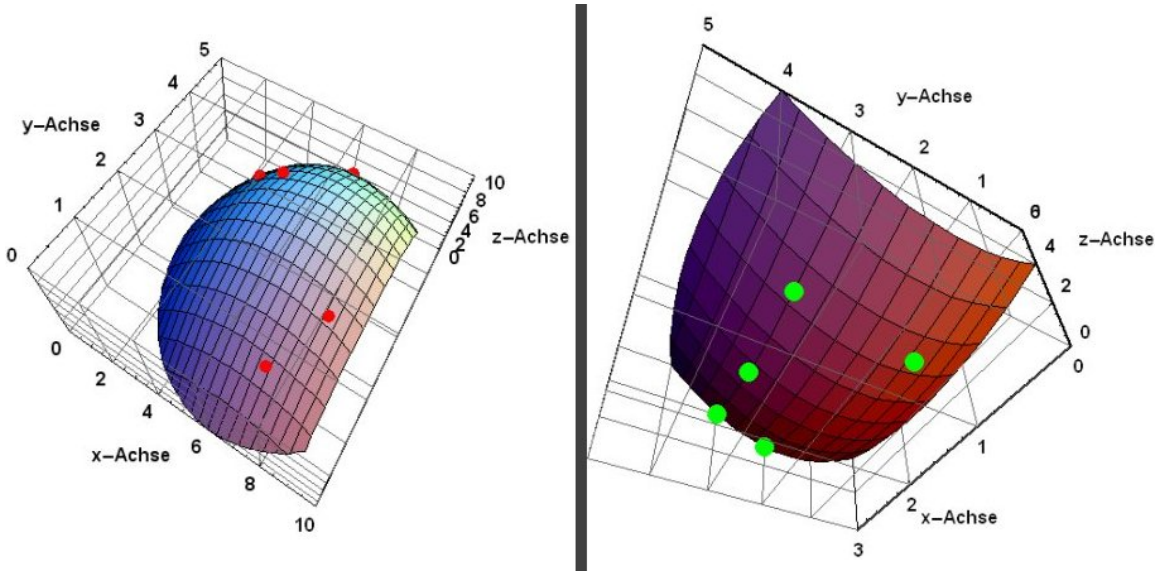


Abb.: 3.4: Hyperflächen für 10 Datensätze mit jeweils 5 Klassen

Die Abb. 3.5 stellt beide Hyperfläche gemeinsam dar. Deutlich erkennt man nun die Überlappung der beiden Hyperflächen. Falls zu klasssifizierende Datensätze in diesen Bereich fallen, ist eine Zuordnung des Datensatzes zu einer Klasse nicht mehr möglich.

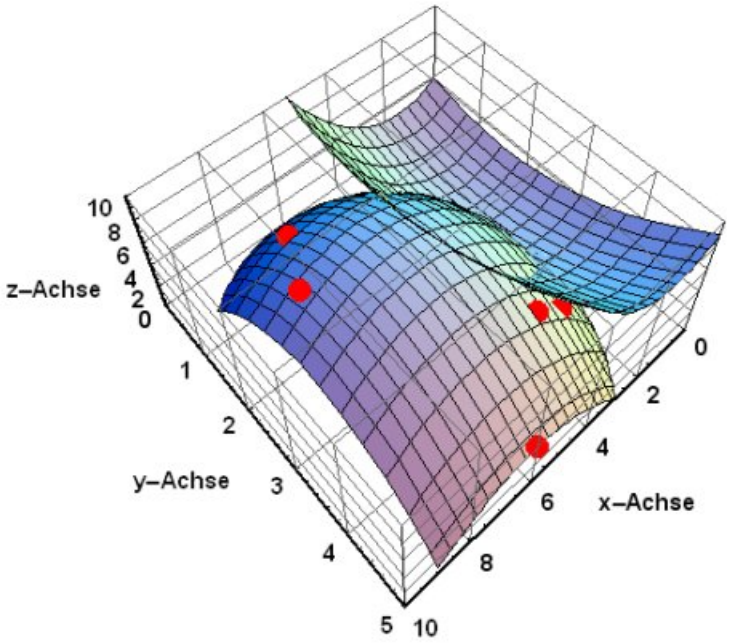


Abb.: 3.5: Überlappende Hyperflächen

Bei konkreten Klassifizierungsaufgaben führt die Vielzahl der Datensätze zu ausgleichenden Hyperflächen, die mitten durch die Punktwolken verlaufen.

Das Auftreten von Überlappungsbereichen hängt nicht nur von der Wahl eines geeigneten mathematischen Modells ab, sondern vor allem von der Lage der Punkte im Hyperraum. Die Abb. 3.6 visualisiert diese Problematik am Beispiel eines linearen mathematischen Modells für je 3 Punkten für die beiden Klassen. Die Abb. 3.6 zeigt die beiden Hyperflächen jeweils in einer eigenen Darstellung. Alle drei Punkte liegen exakt auf der jeweiligen Hyperfläche.

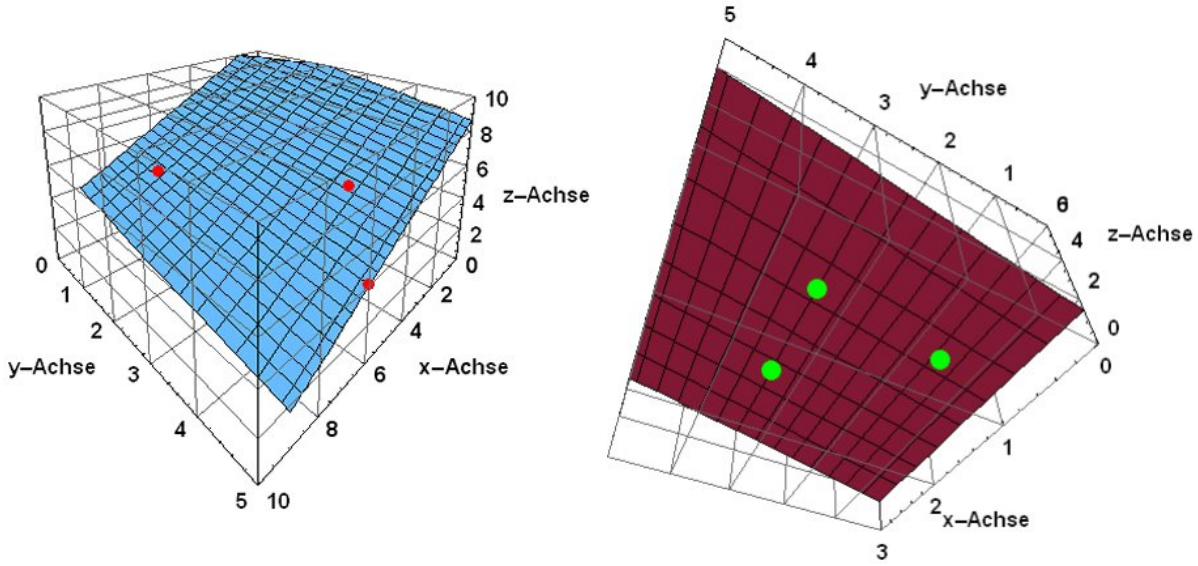


Abb.: 3.6: Darstellung der Hyperflächen für die beiden Klassen

Verschiebt man nun, wie in der Abb. 3.7 in der rechten Darstellung zu erkennen, einen der grünen Punkte, dann ergibt sich gegenüber der gemeinsamen Darstellung der beiden Hyperflächen in der linken Darstellung eine stärker geneigte Hyperfläche für die grünen Punkte und die Hyperflächen besitzen nun keinen Überlappungsbereich mehr.

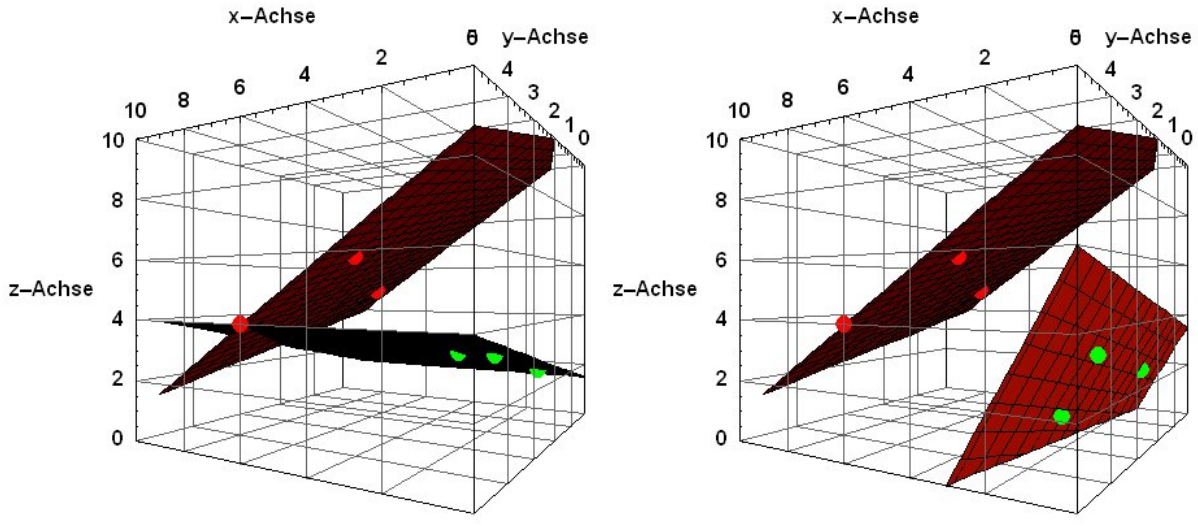


Abb.: 3.7: Überlappende und nicht überlappende Hyperflächen

Alle Punkte lassen sich nun eindeutig trennen, wie in der rechten Darstellung der Abb. 3.7 zu erkennen ist.

Den gleichen Effekt hätte man mit einem anderen mathematischen Modell erzeugen können, das im Bereich der grünen Punkte eine nach unten gekrümmte Fläche generiert.

Für eine leistungsfähige Klassifizierung sollte der Abstand zwischen den Hyperflächen möglichst groß sein bzw. gilt es jene Hyperflächen zu finden, deren minimaler Abstand im Definitionsbereich der Datensätze einen maximalen Wert annimmt.

Beispiel zweier Punktwolken

Die Abb.: 3.8 zeigt zwei Punktwolken, die auf jeweils 50 simulierten Datensätzen mit den Koordinaten (x,y,z) und zwei angenommenen Klassen basieren.

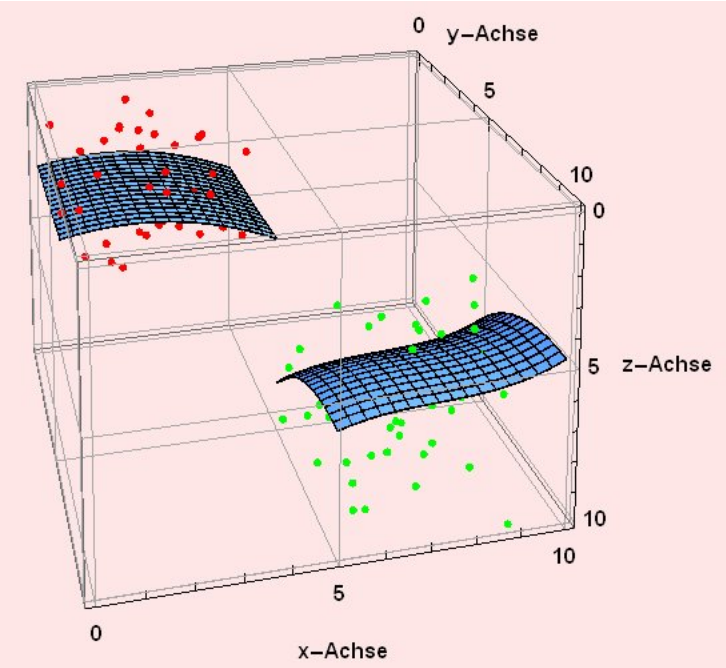


Abb.: 3.8: Beispiel zweier Punktwolken im 3D-Raum

Als mathematisches Modell dient für die erste Klasse mit den roten Punkten ein räumliches quadratisches Polynom (Gl.: 3.16). Für die grünen Punkte der zweiten Kasse wurde ein räumliches kubisches Polynom (Gl.: 3.17) angenommen. Wie die beiden Gleichungen verdeutlichen, stellen die mathematischen Modelle einen funktionalen Zusammenhang zwischen einem einzelnen ausgewählten Attribut, z.B. z, und den restlichen Attributen x und y her. Liegt ein Punkt auf einer der beiden Hyperflächen, dann ist die entsprechende Gleichung erfüllt.

$$z(x,y) = a_1 + a_2 \cdot x + a_3 \cdot x^2 + a_4 \cdot y + a_5 \cdot y^2 \quad (3.16)$$

$$z(x,y) = b_1 + b_2 \cdot x + b_3 \cdot x^2 + b_4 \cdot x^3 + b_5 \cdot y + b_6 \cdot y^2 + b_7 \cdot y^3 \quad (3.17)$$

Zur eigentlichen Klassifizierung setzt man nur noch den neuen Datensatz mit den Attributen $(x_{neu}, y_{neu}, z_{neu})$ in die Gl.: 3.16 und die Gl.: 3.17 ein und bestimmt den Betrag aus der Differenz der linken und rechten Gleichungsseite. Die Gl.: 3.18 bzw. Gl.: 3.19 beschreiben diese Vorgehensweise.

$$d(a) = \| z_{neu} - a_1 - a_2 \cdot x_{neu} - a_3 \cdot x_{neu}^2 - a_4 \cdot y_{neu} - a_5 \cdot y_{neu}^2 \| \quad (3.18)$$

$$d(b) = \| z(b) - b_1 - b_2 \cdot x_{neu} - b_3 \cdot x_{neu}^2 - b_4 \cdot x_{neu}^3 - b_5 \cdot y_{neu} - b_6 \cdot y_{neu}^2 - b_7 \cdot y_{neu}^3 \| \quad (3.19)$$

Der neue Datensatz gehört dann zu jener Klasse, die den geringeren Differenzwert d aufweist.

Eine Klassifizierung mittels Regression bietet sich vor allem dann an, wenn die Datensätze auf physikalischen oder anderen funktionalen Zusammenhängen basieren. Die explizite Form dieser Zusammenhänge kann durchaus unbekannt sein. Typische Beispiele finden sich im Bereich meteorologischer und geophysikalischer Daten und bei dynamischen Systemen, die auf bestimmten Gesetzmäßigkeiten basieren.

Wertet man jedoch z.B. Fragebogenaktionen aus, befasst sich mit betriebswirtschaftlichen Daten oder denkt an Anwendungen im Bereich medizinischer Daten, dann existieren keine mathematischen Modelle mehr. Für diese Bereiche kommen dann Regressionsverfahren nicht mehr in Betracht, sondern man wendet die im folgenden Kapitel beschriebenen Support Vektor Verfahren an. Der besondere Vorteil dieser Verfahren besteht darin, dass es die Punktwolken nicht mehr, wie in der Abb.: 3.8 dargestellt, durch einzelne Hyperflächen ausgleicht, sondern durch eine ebene Hyperfläche, wie in der Abb.: 3.9 zu erkennen ist, trennt.

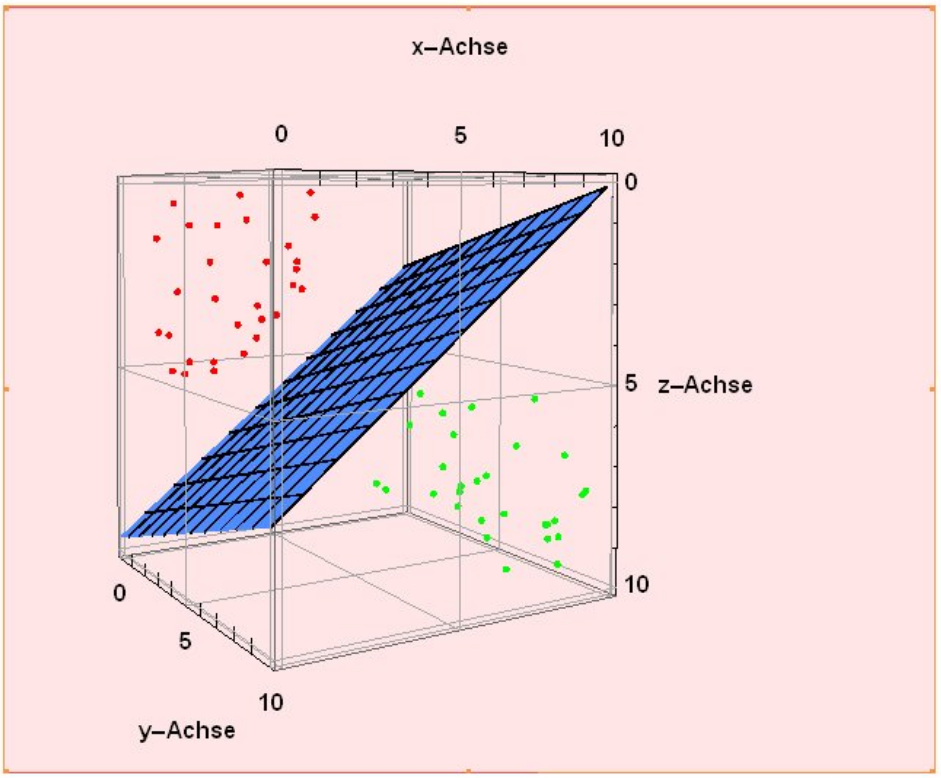


Abb.: 3.9: Trennung zweier Punktwolken durch eine Fläche im 3D-Raum

Liegt ein Punkt links einer Trennfläche, dann gehört er zur einen Klasse und befindet sich ein Punkt rechts der Hyperfläche, dann ordnet man dem Punkt die andere Klasse zu. Statt einem mathematischen Modell, dessen Struktur häufig unbekannt oder gar nicht vorhanden ist, wendet man beim Support Vektor Verfahren „Kernelfunktionen“ an, die allgemein bekannte Strukturen aufweisen.

3.5 Support Vektor Verfahren

Support Vektoren, dessen deutscher Ausdruck „Stützvektoren“ kaum gebräuchlich ist, sind spezielle Vektoren innerhalb des Support Vektor Verfahrens, wobei die Rechenergebnisse von der Lage dieser Support Vektoren abhängen. Je nach dem Typ des vorliegenden Rechenfahrens bezeichnet man die entsprechenden Algorithmen als Support Vektor Maschine, als Support Vektor Regression oder als Support Vektor Klassifikation. Der Ausdruck „Maschine“ weist auf die Herkunft dieses Verfahrens aus dem Gebiet des maschinellen Lernens hin. Die grundlegenden Arbeiten zu Support Vektoren stammten aus dem Jahr 1995 und gehen auf Vapnik [17] zurück.

3.5.1 Trennbare Punktwolken

Die Abb. 3.10 stellt die grundlegende Problematik bei trennbaren Punkten graphisch dar. Die Punkte symbolisieren die Messwerte einer roten und grünen Klasse. Aus Gründen der Anschaulichkeit handelt es sich nur um 2-dimensionale Punkte, die jeweils einen Datensatz mit zwei Attributen visualisieren. Im n-dimensionalen Raum entsprechen die dargestellten Geraden des 2D-Raumes dann Hyperflächen, deren Dimension um einen Grad von n auf n-1 reduziert ist.

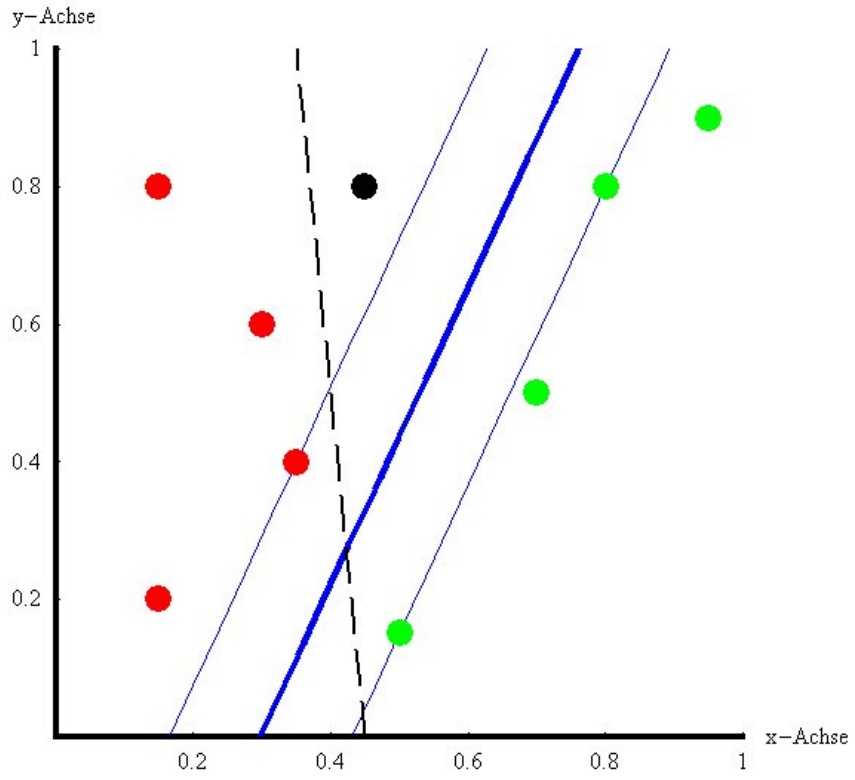


Abb.: 3.10: Beispiel für Support Vektoren

Die Klassifizierung eines neuen Punktes, dessen Klasse nicht bekannt ist, erforderlich eine geometrische Trennung der roten und grünen Punktgruppen durch die durchgezogene dicke blaue Linie. Liegt der neue Punkt links der Trennlinie, dann gehört er zur roten Klasse bzw. umgekehrt zur grünen Klasse. Die beiden parallel verlaufenden dünnen Linien begrenzen einen punktfreien Korridor. Punkte, die auf diesen beiden Grenzlinien liegen, nennt man Support Vektoren.

Grundsätzlich erfordert ein erfolgreiches Trennen einen möglichst breiten freien Raum zwischen den Punktgruppen. Allgemein existieren in der Abb. 3.10 beliebig viele Trennlinien, wie z.B. die schwarze gestrichelte Linie. Für diese Trennlinie misslingt die Klassifizierung des eingezeichneten neuen schwarzen Punktes, denn der entsprechende Datensatz würde der Klasse der roten Punkte zugeordnet. Dagegen ergibt sich im Fall der durchgezogenen blauen Trennlinie die richtige Klassifizierung.

Die Lage und die Breite der Trennfläche hängen von den Punkten ab, die in der Nähe der linken und rechten parallelen Trennlinien liegen. Punkte, die weiter weg von den Trennlinien liegen, befinden sich hinter den Support Vektoren und gehen daher in die Berechnungen kaum ein.

Die mathematische Beschreibung von Support Vektor Verfahren verwendet folgende Begriffe:

Anzahl der Datensätze	n
Anzahl der Attribute	m
Senkrechter Abstand der Trennfläche zum Nullpunkt	b
Lotvektor der Trennfläche	w
einzelner Datensatzvektor i	$x_i = \{x_{i,1}, \cdots, x_{n,m}\}$
Klassenvektor	$y = \{y_1, \cdots, y_n\}; \quad y_i \in \{-1, 1\}$
Gleichung der Trennfläche	$w^T x + b = 0$
Abstand des Punktes x_i von der Trennfläche	$d_i = w^T x_i + b$

(3.20)

Beispiel mit 2D-Punkten

Die Abb. 3.10 enthält die folgenden Parameter und Vektoren:

$$
\begin{aligned}
n &= 8 \\
m &= 2 \\
b &= -0.27 \\
w &= (0.907633, -0.419764)^T \\
dmax &= 0.120234
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= (0.15, 0.2)^T \\
x_2 &= (0.35, 0.40)^T \\
x_3 &= (0.3, 0.6)^T \\
x_4 &= (0.15, 0.8)^T \\
x_5 &= (0.5, 0.15)^T \\
x_6 &= (0.8, 0.8)^T \\
x_7 &= (0.7, 0.5)^T \\
x_8 &= (0.95, 0.9)^T
\end{aligned}
\qquad (3.21)
$$

$$
\begin{aligned}
y &= (1, 1, 1, 1, -1, -1, -1, -1)^T \\
w^T x + b &= 0 \\
d_1 &= w^T x_1 + b = 0.907633 \cdot 0.15 - 0.419764 \cdot 0.20 - 0.27 = -0.217808 < 0 \\
d_8 &= w^T x_8 + b = 0.907633 \cdot 0.95 - 0.419764 \cdot 0.9 - 0.27 = 0.214464 > 0
\end{aligned}
$$

Der Punkt x_1 (links, unten) liegt wegen $d_1 < 0$ links und der Punkt x_8 (oben, rechts) gehört wegen $d_8 > 0$ zur rechten Seite.

Die Trennlinie weist stets eine um eine Stufe reduzierte Dimension auf. Die zweidimensionalen Punkte in der Abb. 3.10 trennt eine Gerade. Für höhere Dimensionen tritt die Trennlinie als Hyperfläche auf. Die Definitionsgleichung einer Hyperfläche in der Hesseschen Normalform lautet:

$$w^T x + b = 0 \tag{3.22}$$

Für eine leistungsfähige Klassifizierung muss der Abstand der beiden Trennflächen ein Maximum annehmen. Gesucht ist somit ein möglichster breiter und punktfreier Korridor. Der Wert des Minimums aller senkrechten Punktabstände zur Hyperfläche dient zur Normierung:

$$\min_i \left[\ |w^T x_i + b| \ \right] = 1 \ ; \quad i = 1, \cdots, n \tag{3.23}$$

Nimmt der minimale Abstand nicht den geforderten Wert von Eins an, dann teilt man die Gl. 3.22 durch den minimalen Abstand. Falls jedoch der minimale Abstand für eine bestimmte Trennflächen den Wert Null erreicht, dann schneidet diese Trennfläche einen Punkt und eine Zuordnung dieses Punktes zu einer Klasse ist nicht mehr möglich. In diesem Sonderfall erfüllt die Hyperfläche nicht die Bedingung einer Trennfläche und wird deshalb verworfen.

Mathematische Vereinfachungen lassen sich durch die Berechnung relativer Abstände erzielen:

$$d(w, b, x) = \frac{|w^T x + b|}{\sqrt{w^T w}} \tag{3.24}$$

Mit der Gl. 3.23 folgt wegen $y_i \in \{-1, 1\}$ ein einfaches Kriterium, ob es sich bei einer angenommenen Hyperfläche tatsächlich um eine Trennfläche handelt. Für alle links von der Hyperfläche liegenden Punkte ergibt das Produkt aus dem dann negativen Abstandswert und dem negativ definierten Wert der Klasse $y = -1$ einen positiven Wert. Falls sich ein Punkt mit der Klasse $y = 1$ jedoch ebenfalls links der Hyperfläche befindet, dann nimmt das Produkt einen negativen Wert an und bei der Hyperfläche handelt es sich um keine Trennfläche. Damit eine Hyperfläche zwei Klassen mit den Werten ± 1 und unter der Bedingung der Gl. 3.23 trennt, gilt für alle Punkte beider Klassen:

$$y_i \cdot \left(w^T x_i + b \right) \geq 1 \tag{3.25}$$

Mit diesen Festlegungen und der Annahme, dass zunächst nur zwei Klassen u,v mit den Werten ± 1 vorliegen, folgt für den zu maximierenden Abstand p zwischen den beiden Klassen:

$$\begin{aligned} \max\left[p(w,b)\right] &= \min_u \left[d(w,b,x_i)\right] + \min_v \left[d(w,b,x_i)\right] \\ &= \min_u \left[\frac{|w^T x + b|}{\sqrt{w^T w}} \right] + \min_v \left[\frac{|w^T x + b|}{\sqrt{w^T w}} \right] \\ &= \frac{1}{\sqrt{w^T w}} \left(\min_u \left[|w^T x + b| \right] + \min_v \left[|w^T x + b| \right] \right) \\ &= \frac{2}{\sqrt{w^T w}} \end{aligned} \tag{3.26}$$

Auf Grund der Normierung tritt der Parameter b in der Gl.: 3.26 nicht mehr auf und fällt daher zunächst aus der Optimierung heraus.

Als weitere Vereinfachung ist es sinnvoll, statt den in der Gl. 3.26 erhaltenen Ausdruck $\frac{2}{\sqrt{w^T w}}$ zu maximieren, besser die inverse Funktion zu minimieren, wobei die Nebenbedingung der Gl. 3.23 zu beachten ist.

$$\min\left[\Phi(w)\right] = \frac{1}{2}\sqrt{w^T w} \quad \text{mit den w-Vektoren einer trennenden Hyperfläche} \qquad (3.27)$$

Diese merkwürdige Minimum-Maximum Berechnung hängt damit zusammen, dass der kleinste Abstand d_{min} den freien Korridor zwischen den Punkten begrenzt und daher zu maximieren ist.

Berechnungsbeispiel einer Trennfläche im 3D-Raum

Ein einfacher Algorithmus zur Trennung der Klassen wählt schrittweise alle möglichen Hyperflächen durch die systematische Variation der Vektoren w und des Skalars b aus und prüft dann mittels der Gl. 3.25, ob es sich bei der vorliegenden Hyperfläche tatsächlich um eine trennende Fläche handelt. Ist dies der Fall, d.h. die gewählte Trennfläche schneidet keinen der Punkte, dann berechnet man für diese Trennfläche mittels der Gl. 3.24 den Abstand zu allen Punkten und normiert die Gleichung der Hyperfläche mit dem minimalen Abstand. Anschließend ist zu prüfen, ob der gefundene minimale Abstand das bisherige Minimum unterschreitet und bei einer positiven Antwort dieser w-Vektor und der entsprechende b-Wert abzuspeichern.

Die Abb. 3.11 stellt zwei simulierte Punktwolken mit je fünf Punkten dar, welche den rot bzw. grün markierten Klassen entsprechen. Das Computerprogramm zur Erzeugung dieser Graphik besteht im wesentlichen aus vier geschachtelten Schleifen für die drei Komponenten wx, wy,wz des w-Vektors und der skalaren Größe b. Das Minimum der w-Werte und somit der maximal breite, punktfreie räumliche Korridor wird durch die dargestellte Fläche geteilt. Es ist deutlich zu erkennen, wie der rote und der grüne Supportvektor die jeweils dazu gehörende Trennflächen „stützen“. Auf Grund der Gleichung 3.22, handelt es sich bei allen Hyperflächen um Ebenen.

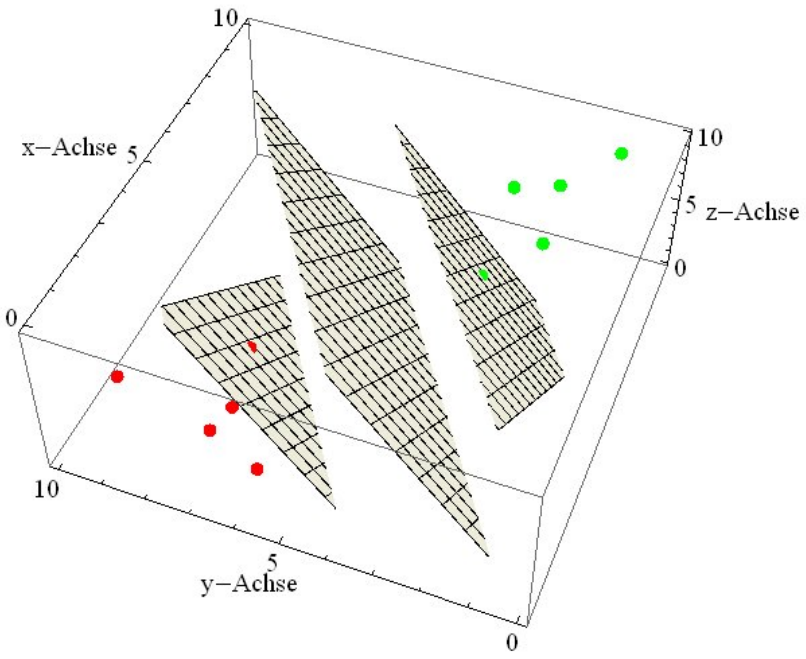


Abb.: 3.11: Beispiel für Support Vektoren

Dieser anschauliche Algorithmus eignet sich nur für Datenbestände mit einer geringen Anzahl von Attributen, denn das erläuterte „brute force“ Verfahren erfordert bei einer größere Anzahl von Attributen zahlreiche geschachelte Schleifen, deren Auswertung zu einem nicht vertretbaren Aufwand an Rechenzeit führen.

3.5.2 Der Kerneltrick bei überlappenden Punktwolken

Bei den bisherigen Beispielen lagen die Punkte in der Ebene immer so günstig verteilt, dass eine lineare Trennung möglich war. Häufig überlappen sich jedoch die Bereiche unterschiedlicher Klassen (Abb.: 3.12) oder die Punkte sind konzentrisch angeordnet . In beiden Fällen existiert dann keine die Punkte trennende Ebene mehr.

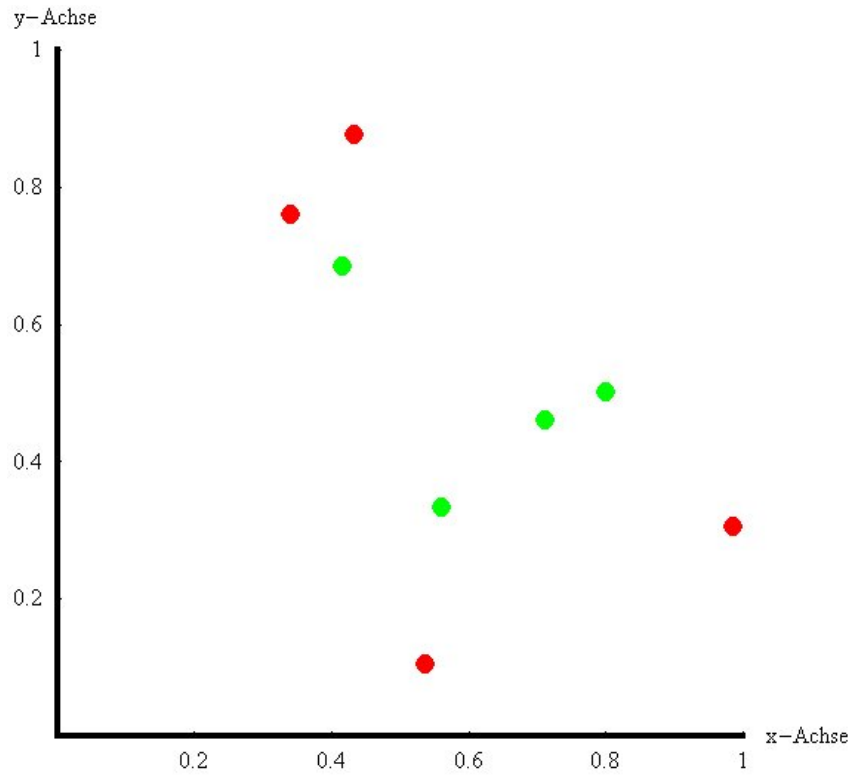


Abb.: 3.12: Punkteverteilung für die keine Trennfläche existiert

Hier hilft nun der so genannte Kerneltrick weiter. Im ersten Schritt erhöht man in Gedanken die Ordnung der Punkte, denn in einem nur genügend hochdimensionalen Vektorraum lassen sich die Punkte durch Hyperflächen trennen. Die dazu erforderlichen Transformationsgleichungen wendet man jedoch nur auf die Berechnung der Skalarprodukte an, denn für die Bestimmung der Abstände einzelner Punkte zur Hyperfläche gemäß der Hesseschen Normalform genügt auch im hochdimensionalen Vektorraum als einziger Algorithmus die Auswertung von Skalarprodukten. Geeignete Transformationsgleichungen, die man in diesem Zusammenhang als Kernelfunktionen bezeichnet, sind:

$$K_l(w,x) = w^T x \quad \text{linear} \tag{3.28}$$
$$K_p(w,x) = (\gamma w^T x + c)^d \quad \text{Polynom} \tag{3.29}$$
$$K_r(w,x) = e^{-\gamma \|w-x\|^2} \quad \text{Radial Basis Funktion} \tag{3.30}$$
$$K_s(w,x) = tanh(w^T q + c) \quad \text{Sigmoide Funktion} \tag{3.31}$$

Die Wahl der Parameter γ und c legen die Ordnung des neuen Hyperraumes fest. Durch die Kombination unterschiedlicher Kernelfunktionen entstehen neue Kernelfunktionen, wie z.B.:

$$K_1(w,x) = w^T x + (\gamma w^T x + c)^d \quad \text{Summe} \tag{3.32}$$
$$K_2(w,x) = w^T x \cdot (\gamma w^T x + c)^d \quad \text{Produkt} \tag{3.33}$$

Beispiel

Gegeben sei der zwei-dimensionale Vektor $x = (0.2, 0.5)^T$ und eine Abbildung

$$(x_1, x_2)^T \quad \rightarrow \quad (x_1 \cdot x_1,\ \sqrt{2} \cdot x_1 \cdot x_2,\ x_2 \cdot x_2) \tag{3.34}$$

Damit erweitert sich der Vektor x um eine Dimension auf den Vektor z:

$$x = \begin{pmatrix} 0.2 \\ 0.5 \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} 0.2 \cdot 0.2 \\ 2 \cdot 0.2 \cdot 0.5 \\ 0.5 \cdot 0.5 \end{pmatrix} = \begin{pmatrix} 0.04 \\ 0.2 \\ 0.25 \end{pmatrix} \tag{3.35}$$

Beispiel für eine einfache Kernelfunktion

Wendet man den Kerneltrick auf das in der Abb. 3.12 dargestellte Beispiel an und wählt zur Transformation der Punkte in den 3D-Raum wieder den linearen Zusammenhang gemäß der Gl. 3.34, dann gelingt die fehlerfreie Klassifikation der 8 Punkte. In der Abb. 3.13 erkennt man, dass die roten bzw. grünen Punkte nun auf den gegenüber liegenden Seiten der Hyperfläche erscheinen.

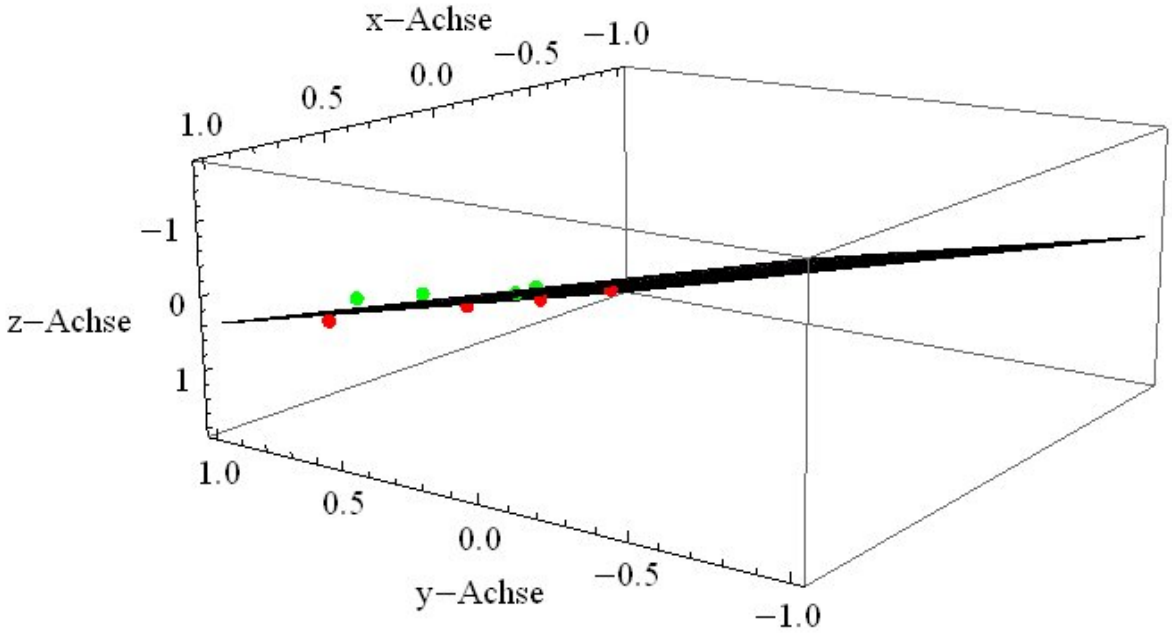


Abb.: 3.13: Vorder- und Rückansicht der Punkteverteilung im 3D-Raum

Allerdings ergibt sich eine maximale Breite des trennenden Korridors von nur noch $d_{max} = 0.05$, was bezogen auf den Wertebereich der Punkte 5% entspricht. Erweitert man den Vektorraum dagegen auf 5 Dimensionen, dann erhält man mit der Transformationsgleichung

$$(x_1, x_2)^T \quad \rightarrow \quad (1, x_1^2, \sqrt{2}\ x_1 x_2, x_2^2, \sqrt{2}\ x_1, \sqrt{2}\ x_2) \tag{3.36}$$

eine maximale Breite von ca. 12%. Allerdings steigt dabei die Rechenzeit enorm an, denn nun sind 5 Schleifen für die Komponenten des w-Vektors und eine Schleife für die b-Werte zu durchlaufen. Als Ergebnis folgt dann eine 4D-Hyperfläche.

Der besondere Vorteil der gewählten 5D-Transformation besteht darin, dass sich beim Bilden von Skalarprodukten viele Terme zusammenfassen lassen und somit im 5D-Raum die einfache Rechenregel gemäß der Kernelfunktion 3.29 für die Parameter $\gamma = c = d = 1$ gilt:

$$x^T x_i = (1 + x^T x_i) \tag{3.37}$$

Die fehlerfreie Klassifizierung von Daten in einem Raum höherer Dimension stellt ein interessantes Verfahren dar, das 1958 von Frank Rosenblatt, von dem auch das Perzeptron eines neuronalen Netzes stammt, veröffentlicht wurde. Wegen der relativ großen Anforderung an die benötigte Rechenleistung gelang diesem Verfahren erst vor ca. 10 Jahren der Durchbruch. Nachteilig an dem Verfahren ist allerdings die Wahl einer geeigneten Kernelfunktion sowie deren Parameter, die von der Lage der gegebenen Punkte abhängen. Theoretisch muß man nur die Dimension immer weiter erhöhen und erhält dann eine fehlerfreie Trennung der Punkte, wobei der dazu erforderliche Rechenaufwand ebenfalls mit der Dimension in extremer Weise ansteigt.

Die ungünstige Lage einzelner Punkte im Vektorraum kann unter Umständen jede Klassifizierung verhindern, weil die Trennung erst in extrem hochdimensionalen Räumen gelingen würde. Trotzdem akkzepiert man beim Lagrange-Ansatz alle gegebenen Punkte, weist ihnen jedoch eine Gewichtung zu. Beim Test, ob eine bestimmte Hyperfläche auch tatsächlich alle Punkte trennt, berechnet man für einen auf der falschen Seite liegenden Punkt den Abstand und subtrahiert ihn vom Betrag des w-Vektors. Die zu minimierende Kostenfunktion L lautet dann:

$$L(w,b,\alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \tag{3.38}$$

Bei (w,b) handelt sich um eine spezielle Hyperfläche, die den minimalen Abstand zu einem Supportvektor von exakt Eins besitzt. Die Normierungskonstante ergibt sich aus allen absoluten Abständen und entspricht dem minimalen Wert dieser Abstände. Der Support Vektor leistet somit keinen Beitrag zur Kostenfunktion L.

Punkte, die auf der falschen Seite der Hyperfläche liegen, erhöhen den Wert von L und umgekehrt reduzieren die Punkte auf der richtigen Seite die Kostenfunktion. Mit dem Faktoren α_i „bestraft“ bzw. „belohnt“ man die Lage der einzelnen Punkte. In der Kostenfunktion L kommt es darauf an, die Werte für (w,b) zu minimieren und für die Gewichte α_i zu maximieren. Insgesamt liegt hier eine Optimierungsaufgabe vor, die sich durch Nullsetzen der partiellen Ableitungen der Kostenfunktion L lösen lässt.

$$\frac{\delta}{\delta b}L(w,b,\alpha) = 0 \quad \rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3.39}$$

$$\frac{\delta}{\delta w}L(w,b,\alpha) = 0 \quad \rightarrow \quad w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{3.40}$$

Setzt man diese Ausdrücke in die Kostenfunktion ein, dann erhält man nach einigen Umformungen eine neue Funktion, eine Gewinnfunktion, die nun zu maximieren ist. Dieses Lösungsschema ist in der Literatur als „duales Problem“ bekannt. Die Lösung des dualen Problems liefert einzig die Werte für α_i, mit denen sich der größte Wert der Gewinnfunktion W einstellt und somit auch ein maximal breiter Trennbereich.

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{n} \alpha_i \; \alpha_j \; y_i \; y_j \; (x_i^T x_j)\right) \tag{3.41}$$

$$\alpha_i \geq 0 \tag{3.42}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3.43}$$

Die Lösung dieser Optimierungsaufgabe ergibt positive Werte für die gesuchten α-Werte, wobei die Anzahl der α-Werte der Anzahl der Datenpunkte entspricht.

Mit den optimalen α-Werten folgt dann der entsprechende Normalenvektor w und die Breite des Intervalles d_{max} zu:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{3.44}$$

$$dmax = \frac{2}{w^T w} \tag{3.45}$$

Schließlich fehlt noch der Wert für den Offset b, den man als Mittelwert aus allen Punkten der Trainingsmenge bestimmt:

$$\alpha_i \left[y_i \left(\sum_{j=1}^{n} \alpha_j (y_j x_i) + b \right) - 1 \right] = 0 \tag{3.46}$$

$$b_i = y_i - \sum_{j=1}^{n} \alpha_j (y_j x_i) \tag{3.47}$$

$$b = \frac{1}{n} \sum_{j=1}^{n} b_j \tag{3.48}$$

Die Klassifizierung neuer Punkte erfordert dann die Berechnung des Vorzeichens des Abstandes, den dieser Punkt von der optmalen Hyperfläche entfernt liegt:

$$\text{Klasse}(x) = \text{sign} \left(\sum_{i=1}^{n} \alpha_i y_i x_i^T x + b \right) \tag{3.49}$$

An den Gleichungen zur Berechnung der α-Werte, des Offsets b und der Abstände (Gl. 3.49) lässt sich nun der „Kerneltrick“ gut erkennen, denn die Gleichungen enthalten immer nur das Skalarprodukt der Punkte. Die Kernelfunktionen gemäß den Gl.: 3.28 bis Gl.: 3.33 ersetzen dieses Skalarprodukt.

Dadurch entfällt auf verblüffende Weise die explizite Berechnung der w-Vektoren und somit sind auch keine Transformationsgleichungen erforderlich.

Basierend z.B. auf einem Polynom-Kernel, berechnet man das Skalarprodukt zwischen den beiden Vektoren u,v wie folgt:

$$(u^T v) \quad \Longrightarrow \quad (u^T v + 1)^5 \tag{3.50}$$

Eine weitere hilfreiche Möglichkeit zur optimalen Einstellung des Support Vektor Verfahrens besteht in der Wahl eines weiteren Parameters c, der beide Summen in der Gl. 3.41 unterschiedlich gewichtet.

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{c}{2} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_i \ \alpha_j \ y_i \ y_j \ (x_i^T x_j) \right) \tag{3.51}$$

Die vielfältigen Einstellmöglichkeiten des Support Vektor Verfahrens ermöglichen sehr genaue Ergebnisse. Allerdings erfordert die nicht-lineare und mit den Nebenbedingungen (Gl. 3.42, 3.43) versehene Optimierungsaufgabe (Gl. 3.41) aufwendige numerische Lösungsverfahren, die für den Einsatz in der beruflichen Praxis kaum geeignet sind.

3.5.3 Least Square Support Vektor Maschinen

Least Square Support Vektor Maschinen berechnen auf elegante und numerisch stabile Weise die Hyperflächen von Support Vektor Maschinen. Im Jahr 1998 entstanden die ersten grundlegenden Ansätze dieses numerischen Lösungsverfahrens, das mittels eines Näherungsansatzes das ursprüngliche nicht-lineare und mit den Nebenbedingungen Gl. 3.53 bzw. Gl. 3.54 behaftete und numerisch schwer zu lösende Optimierungsproblem (Gl. 3.52)

$$W(\alpha) = \text{Maximum}\left[\sum_{i=1}^{n}\alpha_i - \frac{c}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\alpha_i\ \alpha_j\ y_i\ y_j\ (x_i^T x_j)\right)\right] \tag{3.52}$$

$$\alpha_i \geq 0 \tag{3.53}$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{3.54}$$

nun als Regresssionsaufgabe formuliert und am Ende das Lösungsverfahren auf eine Matrizeninversion reduziert. Das Fachbuch [23] beschreibt eine Reihe von Anwendungen für dieses Verfahren.

Der entscheidende Ansatz des numerischen Lösungsverfahrens besteht in der Näherung, die Ungleichung der Abstandsformel durch eine Gleichung zu ersetzen und zum Ausgleich Schlupfvariable ξ anzusetzen.

$$y_i \cdot (w^T x_i + b) < 1 \quad \longrightarrow \quad y_i \cdot (w^T x_i + b) = 1 - \xi; \quad i = 1, \cdots, M \tag{3.55}$$

Die Forderung nach minimalen Werten für die Schlupfvariablen führt dann zur neuen Lagrange-Gleichung, welche den Bereich zwischen den beiden Punktwolken maximiert:

$$Q(w, b, \alpha, \xi) = \frac{1}{2}w^t w + \frac{c}{2}\sum_{i=1}^{M}\xi_i^2 - \sum_{i=1}^{M}\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) \tag{3.56}$$

Das partielle Differenzieren der Lagrange-Gleichung 3.56 und das Nullsetzen der entsprechenden Ableitungen liefert drei Bestimmungsgleichungen zur Minimierung von Q:

$$\frac{\delta Q}{\delta w} \quad \longrightarrow \quad w = \sum_{i=1}^{M}\alpha_i y_i x_i \tag{3.57}$$

$$\frac{\delta Q}{\delta b} \quad \longrightarrow \quad 0 = \sum_{i=1}^{M}\alpha_i y_i \tag{3.58}$$

$$\frac{\delta Q}{\delta \xi} \quad \longrightarrow \quad \alpha_i = C\xi_i \tag{3.59}$$

Die Gl. 3.59 führt wegen des normierten Definitionsbereiches von α zu einer Abschätzung der α-Werte:

$$\|\alpha_i\| \leq 1 \quad \longrightarrow \quad 0 \leq \alpha_i \leq c; \qquad i = 1, \cdots, M \tag{3.60}$$

Ein kurzes Beispiel soll die Formelschreibweisen verdeutlichen.

Beispiel zu den Formelschreibweisen für 3 Datensätze

Das Gleichungssystem 3.55 lautet in der für $M = 3$ ausgeschriebenen Form:

$$y_1(w^T x_1 + b) = 1 - \xi_1 \tag{3.61}$$
$$y_2(w^T x_2 + b) = 1 - \xi_2 \tag{3.62}$$
$$y_3(w^T x_3 + b) = 1 - \xi_3 \tag{3.63}$$

Für die Gleichungen 3.57 zur Bestimmung des w-Vektors gilt:

$$w = \alpha_1 y_1 x_1 + \alpha_2 y_2 x_2 + \alpha_3 y_3 x_3 \tag{3.64}$$

Die mit dem Klassenparameter y gewichtete Summe der α-Werte gemäß der Gl. 3.58 lässt sich schreiben:

$$\alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_3 = 0 \tag{3.65}$$

Die Gl. 3.66 besteht aus M einzelnen Gleichungen:

$$\alpha_1 = C \cdot \xi_1 \tag{3.66}$$
$$\alpha_2 = C \cdot \xi_2 \tag{3.67}$$
$$\alpha_3 = C \cdot \xi_3 \tag{3.68}$$

Ersetzt man nun in den Abstandsgleichungen 3.61 die Ausdrücke w, α und ξ durch die Gl. 3.57, 3.65 und 3.66, dann entsteht ein überschaubares Gleichungssystem,

$$y_1(\alpha_1 y_1 x_1^t x_1 + \alpha_2 y_2 x_2^t x_1 + \alpha_3 y_3 x_3^t x_1 + b) = 1 - \frac{\alpha_1}{c} \tag{3.69}$$
$$y_2(\alpha_1 y_1 x_1^t x_2 + \alpha_2 y_2 x_2^t x_2 + \alpha_3 y_3 x_3^t x_2 + b) = 1 - \frac{\alpha_2}{c} \tag{3.70}$$
$$y_3(\alpha_1 y_1 x_1^t x_3 + \alpha_2 y_2 x_2^t x_3 + \alpha_3 y_3 x_3^t x_3 + b) = 1 - \frac{\alpha_3}{c} \tag{3.71}$$
$$\tag{3.72}$$

für das sich eine Matrizenschreibweise anbietet.

$$\begin{pmatrix} y_1 y_1 x_1^t x_1 + \frac{1}{c} & y_1 y_2 x_2^t x_1 & y_1 y_3 x_3^t x_1 \\ y_2 y_1 x_1^t x_2 & y_2 y_2 x_2^t x_2 + \frac{1}{c} & y_2 y_3 x_3^t x_2 \\ y_3 y_1 x_1^t x_3 & y_3 y_2 x_2^t x_3 & y_3 y_3 x_3^t x_3 + \frac{1}{c} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{3.73}$$

Wie aus diesem Beispiel zu erkennen ist, empfiehlt es sich, nun für die weiteren Berechnungen die einfachen Vektor- und Matrix-Schreibweisen zu wählen. Dazu bezeichnet man die quadratische Matrix in der Gl. 3.73 als Ω und interpretiert die Reihe der Einsen als Vektor 1. Mit den weiteren Vektoren für α, w und ξ lassen sich dann auf elegante Weise alle zur Berechnung der Hyperflächen erforderlichen Gleichungen als lineares Gleichungssystem formulieren.

$$\Omega\alpha + yb = 1 \tag{3.74}$$
$$y^T\alpha = 0 \tag{3.75}$$

Die Lösung dieses Gleichungssystems erfordert die Inversion der quadratischen Matrix Ω. Mit algebraischen Umformungen entstehen schließlich die vektoriellen Lösungsgleichungen, wobei zunächst der Skalar b (Gl. 3.77) und dann der Vektor α (Gl. 3.76) zu berechnen ist.

$$\alpha = \Omega^{-1}(1 - yb) \tag{3.76}$$
$$y^T\Omega^{-1}(1 - yb) = 0 \quad \longrightarrow \quad y^T\Omega^{-1}1 - y^T\Omega^{-1}yb = 0$$
$$b = \frac{y^T\Omega^{-1}1}{y^T\Omega^{-1}y} \tag{3.77}$$

Mit den Gl. 3.76 und Gl. 3.77 liegen nun zwei Lösungsgleichungen vor, die sich jedoch numerisch durch die Einführung von Dreiecksmatrizen noch erheblich verbessern lassen. Dazu verwendet man einen Hilfsvektor a und dazu eine geeignete Bestimmungsgleichung, die wegen der quadratischen und symmetrischen Ω-Matrix leicht transponierbar ist.

$$\Omega a = y \quad \text{bzw.} \quad y^T = \Omega^T a^T \tag{3.78}$$

Setzt man die Bestimmungsgleichung Gl.: 3.78 in den Ausdruck zur Berechnung des Skalars b (Gl. 3.77) ein,

$$b = \frac{a^T \Omega \Omega^{-1} 1}{a^T \Omega \Omega^{-1} y} \quad \longrightarrow \quad b = \frac{a^T 1}{a^T y} \tag{3.79}$$

dann entsteht eine neue Bestimmungsgleichung für b, die allerdings noch die Bestimmung des Hilfsvektors a erfordert.

Zur Ermittlung des α-Vektors wählt man einen Hilfsvektor g, der wie folgt definiert ist

$$\Omega g = 1 \quad \longrightarrow \quad g = \Omega^{-1} 1 \tag{3.80}$$

und setzt diesen Hilfsvektor g, ebenso wie den Hilfsvektor a, in die Bestimmungsgleichung für den Vektor α ein.

$$\alpha = g - \Omega^{-1} y b = g - \Omega^{-1} \Omega a b = g - ab \tag{3.81}$$

Zur Berechnung der gesuchten Werte für b und α mittels der Gl. 3.79 bzw. der Gl. 3.81 bedarf es nun zunächst der Bestimmung der Hilfsvektoren a und g, die sich jedoch mittels der Dreieckszerlegung der symmetrischen Ω-Matrix in eine obere Ω_{oben} und in eine untere Ω_{unten} Dreiecksmatrix und deren Inversen, schnell und stabil berechnen lassen.

$$\Omega = \Omega_{oben}\ \Omega_{unten} \tag{3.82}$$

$$a = \Omega_{oben}^{-1}\ \Omega_{unten}^{-1}\ y \tag{3.83}$$

$$g = \Omega_{oben}^{-1}\ \Omega_{unten}^{-1}\ 1 \tag{3.84}$$

Mit den beim Programmstart zu bestimmenden Werten für b und α, basiert dann bei der eigentlichen Klassifizierung die Feststellung der Klasse wieder mit auf der Gl. 3.85

$$\text{Klasse}(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i x_i^T x + b \right) \tag{3.85}$$

Allen Bestimmungsgleichungen gemeinsam ist die erstaunliche Tatsache, dass bezüglich der Vektoren immer nur Skalarprodukte auftreten, die sich durch eine Kernel-Funktion ersetzen lassen.

Das mehrstufige Klassifizierungsverfahren der Least Square Support Vektor Maschinen beginnt zunächst mit der Festlegung der Systemparameter.

- Wahl der Kernel-Funktion und deren interne Parameter
- Gewichtung der Bedingung des punktfreien Korridors durch den Parameter c

Anschließend erfolgt die algorithmische Berechnung des b-Wertes und des α-Vektors.

1. Aufstellen der Ω-Matrix unter Verwendung der Kernel-Funktion
2. Zerlegen der Ω-Matrix in obere und untere Dreiecksmatrizen
3. Berechnen der Hilfsvektoren a und g
4. Berechnen von b und α

Dann erfolgt der Test der eingestellten bzw. berechneten Parameter durch die Klassifizierung von Testdaten, deren bekannte Klassen als Referenz dienen und die zur Berechnung der Systemparameter nicht verwendet wurden. Diese Testdaten kann man sich dadurch beschaffen, dass man die gegebenen Datenbestand in zwei Teile zerlegt. Mit dem ersten und größeren Teil berechnet man die Parameter der Support Vektor Maschine und mit dem zweiten und kleineren Teil führt man den Test der Algorithmen durch. Bei einer Aufschaltung der Trainingsdaten auf die Support Vektor Maschine muss bei einer richtig gewählten Kernelfunktion eine fehlerfreie Klassifizierung der Trainingsdaten erfolgen.

Falls zu viele falsche Klassenzuordnungen auftreten, beginnt man mit einer Anpassung der Kernelfunktion und des Parameters c. Als hilfreich erweisen sich bei diesen Testläufen folgende Fehlerkriterien, die für zwei Klassen A und B wie folgt lauten:

$$R_A = \text{richtig klassifizierte A-Klasse} \tag{3.86}$$
$$F_A = \text{falsch klassifizierte A-Klasse} \tag{3.87}$$
$$R_B = \text{richtig klassifizierte B-Klasse} \tag{3.88}$$
$$F_B = \text{falsch klassifizierte B-Klasse} \tag{3.89}$$

$$S = \frac{R_A}{R_A + F_B} \tag{3.90}$$

$$V = \frac{F_A}{F_A + R_B} \tag{3.91}$$

$$S = ROC(V) \tag{3.92}$$

Die graphische Darstellung der Fehlerkriterien S in Abhängigkeit von V bezeichnet man aus historischen Gründen als ROC (receiver-operating characteristic).

Support Vektor Maschinen arbeiten optimal bei einer Klassifizierung nach zwei Klassen. Weist der Datenbestand mehr Klassen auf, dann bestehen zwei grundsätzliche Möglichkeiten der Erweiterung des Support Vektor Verfahrens auf Multiklassensysteme.

1. Man bildet alle möglichen Klassenpaarungen, wobei sich bei einer Anzahl von K Klassen insgesamt $0.5 \cdot K \cdot (K-1)$ Paarungen ergeben. Nimmt man z.B. drei Klassen A,B,C an, dann folgen die Paarungen A/B, A/C, B/C.
2. Man klassifiziert eine Klasse gegenüber den restlichen Klassen, was zu K Paarungen führt. Nimmt man wieder drei Klassen A,B,C an, dann entstehen die Paarungen A/BC, B/AC und C/AB.

Die endgültige Zuordnung der Klasse eines Datensatzes erfolgt dann im einfachsten Fall durch ein Votingverfahren oder durch die Anwendung eines ergänzenden Klassifizierungsverfahren, wie z.B. einem neuronalen Netz oder einer Fuzzy-Logik. Eine fehlerfreie Klassifizierung liefert im günstigsten Fall für alle Paarungen widerspruchsfreie Aussagen.

Beispiel einer Klassifizierung nach drei Klassen

Die Tab.: 3.17 enthält 17 Datensätze, die den drei Klassen A,B und C zugeordnet sind. Die geringe Anzahl von nur zwei Attributen ermöglicht die graphische Darstellung der Datensätze in Form von farbigen Punkten. In der Abb.: 3.14 stellen die roten Punkte die A-Klasse, die grünen Punkte die B-Klassen und die blauen Punkte die C-Klassen dar. Die punktfreien Korridore erkennt man an den farbigen Geraden, die gemäß dem Support Vektor Verfahren dicht an den jeweiligen Supportvektoren liegen. In der Abb.: 3.14 erkennt man die drei möglichen Paarungen A/BC, B/AC und C/AB.

Datensatznummer	1. Attribute	2. Attribut	Klasse
1	1	6	A
2	2	7	A
3	1	7	A
4	3	7	A
5	1	9	A
6	1	5	A
7	7	8	B
8	8	8	B
9	9	7	B
10	9	9	B
11	7	9	B
12	6	1	C
13	7	2	C
14	8	3	C
15	8	2	C
16	9	1	C
17	7	3	C

Tab.: 3.17: Trainingdaten eines Multiklassensystems

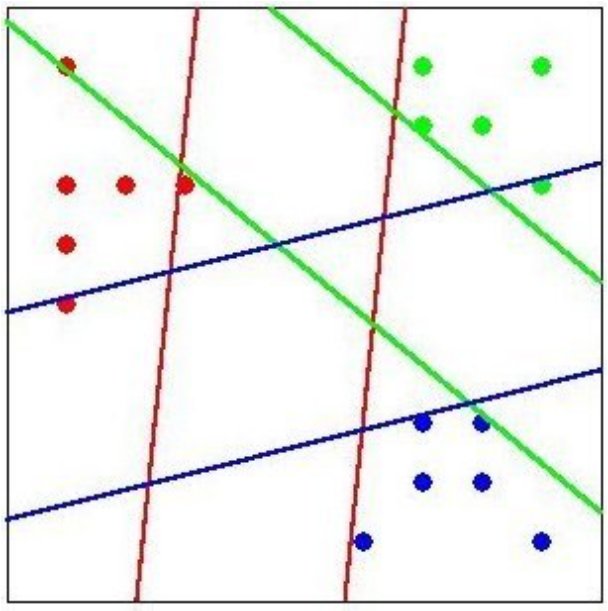


Abb.: 3.14: Lage der Punkte und der punktfreien Korridore

Für jeden der insgesamt 17 Punkte liefern die dreifach auszuführenden Support Vektor Verfahren dann auch drei Abstände. Die Klassifizierung gilt dann als widerspruchsfrei, wenn ein Abstand negativ und die beiden restlichen Abstände positive Werte annehmen. Ein negativer Abstand gehört zu einem Punkt, der links von der entsprechenden Trennlinie liegt, wobei die Trennlinien in einer definierten Richtung verlaufen. In der Abb.: 3.14 führen die roten Linien von oben nach unten und die grünen und blauen Linien von unten nach oben.

Die Tab.: 3.18 listet eine Anzahl von 5 Testdatensätzen auf.

Datensatznummer	1. Attribute	2. Attribut	Klasse
1	5	6	1
2	8	7	2
3	3	6	1
4	5	3	3
5	1	2	1

Tab.: 3.18: Testdaten eines Multiklassensystems

Die Abb.: 3.15 stellt die Lage dieser Punkte in Form von quadratischen Markierungen dar, dessen Farbe der Klasse dieses Punktes entspricht.

Die Datensätze 2,3 und 4 liegen eindeutig in den Bereichen der Trainingsdaten. Daher fallen die entsprechenden Abstände eindeutig aus, d.h. es ergeben sich jeweils ein negativer und zwei positive Abstände.

Der Datensatz 1 wurde absichtlich in das Zentrum der Trennlinien gelegt. Die drei berechneten positiven Abstände verhindern eine eindeutige Klassifizierung. Allerdings kann man sich auf den geringsten Abstand von d=4 zur roten Trennlinie beziehen und näherungsweise die Klasse A zuordnen.

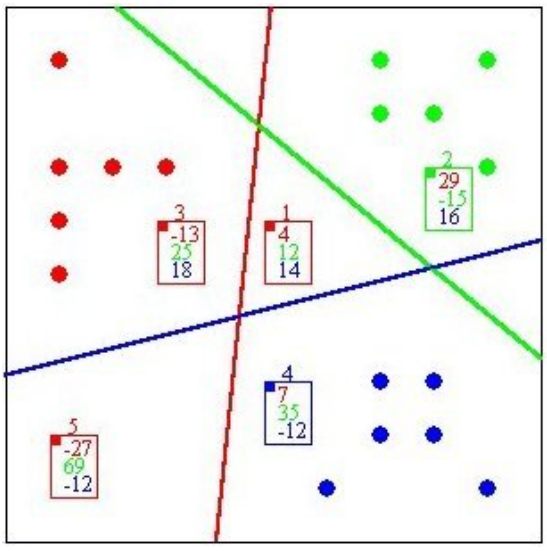


Abb.: 3.15: Lage der Testpunkte

Eine problematische Situation entsteht durch die gewählte Lage des 5. Punktes. Optisch betrachtet könnte er sowohl der roten als auch der blauen Klasse angehören, was die entsprechenden negativen Abstände d=-27 bzw. d=-12 auch bestätigen. Daher kann in diesem Fall nur ein nachgeschaltetes zusätzliches Klassifikationsverfahren eine eindeutige Zuordnung dieses Datensatzes zu einer Klasse garantieren.

3.6 Neuronale Netze

Neuronale Netze stellen einen alternativen Ansatz dar und erfordern weder Informationsberechnungen noch mathematische Modelle. Allerdings besitzen neuronale Netze ihre eigene Problematik, die sich vor allem an der Auswahl einer geeigneten Netzstruktur orientiert. Grundsätzlich lassen sich drei Basisttypen unterscheiden.

1. Vorwärts gekoppelte Netze (Perceptron Netz)
2. Netze mit Rückkopplungen (Hopfield Netz)
3. Feature Maps (Kohonen Netz)

Zu jeder Netzstruktur gehören spezielle Trainingsalgorithmen, wie z.B. der bekannte Backpropagation-Algorithmus, der sich auf vorwärts gekoppelten Netze beschränkt und der als Optimierungsalgorithmus im Falle von Nebenminima zu Konvergenzproblemen führen kann. Hinzu kommt, dass Perceptron-Netze wegen der mangelnder Rückkopplung weniger leistungsfähig sind.

Feature Maps eignen sich besonders für das unüberwachte Lernen. Bei der Klassifizierung liegt jedoch der Fall eines überwachten Lernens vor, denn die Datensätze der Trainingsdaten enthalten als bekannte Größe die Klasse.

Netze mit Rückkopplungen stellen im Bereich der Klassifizierung eine sachgerechte Entscheidung dar. Der entscheidende Vorteil des Hopfield-Netzes kommt hinzu: Die Gewichte eines Hopfield-Netze sind geschlossen und ohne Konvergenzprobleme durch die Auswertung einer Summenformel berechenbar.

Kennzeichnend für ein Hopfield-Netz ist der etwas höhere Rechenaufwand durch das rekursive Auswerten dieses Netzes bei der Klassifikation eines einzelnen Datensatzes. Die Rechenzeiten für die Auswertung des Hopfield-Netzes liegen im Bereich einiger Sekunden, während die alternativen Methoden der Entscheidungsbäume und der Support Vektor Maschinen weit weniger als eine Sekunde Rechenzeit erfordern.

Das Hopfield-Netz besteht aus N Neuronen für die N Attribute eines Datensatzes, N^2 symmetrischen Gewichten, N Eingängen bzw. N Ausgängen und ist mit Ausnahme der Selbstrückkopplung vollständig verbunden. Der Input-Gewichtsfaktor wird konstant auf Eins gesetzt. Die Gewichte sind symmetrisch, denn in der Gleichung (3.96) ist die Reihenfolge der Multiplikation vertauschbar.

$$\text{Anzahl der Neuronen} = \text{Anzahl der Attribute} = N \tag{3.93}$$

$$\text{Anzahl der Datensätze} = M \tag{3.94}$$

$$\text{Gewichte im input Kanal} = 1 \tag{3.95}$$

$$w_{i,j} = \frac{1}{M} \sum_{m=1}^{m=M} (Datensatz_{m,i} \cdot Datensatz_{m,j}) \quad \text{mit } w_{i,j} = w_{j,i} \tag{3.96}$$

$$\text{Anzahl der Gewichte insgesamt} = N(N-1) \tag{3.97}$$

Durch diese Bedingungen erhält man ein Netz, das mittels der Gleichung (3.96) eine geschlossene Berechnung der Gewichte ermöglicht und damit jedes Konvergenzproblem vermeidet. Die Auswertung der Gleichung (3.96) erfolgt einmalig beim Programmstart, was Echtzeit-Anwendungen unterstützt. Allerdings dauert es nach dem Anlegen der Eingangssignale mehrere Schritte bis das Hopfield-Netz einen stabilen Zustand erreicht, denn die rückgekoppelten Ausgangssignale der einzelnen Neuronen beeinflussen im nächsten Schritt wieder das eigene Ausgangssignal. In diesem Zusammenhang ist es günstig, bei der Berechnung der Ausgangssignale abwechselnd nach dem Zufallsprinzip die einzelnen Neuronen auszuwählen und dafür das Ausgangssignal zu berechnen.

3.6.1 Allgemeines Hopfield-Netz

Die Abb.: 3.16 visualisiert z.B. für einen Datenbestand mit drei Attributen das entsprechende neuronale Netz. Zu jedem Attribut gehört ein eigenes Neuron. Legt man an das trainierte Netz einen bestimmten Datensatz aus der Trainingsmenge an, dann liefert das neuronale Netz identische Signale auf den Ausgangskanälen. Das Fehlerminimum, das sich aus dem Vergleich mit allen Trainingsdatensätzen ergibt, nimmt in diesem Testfall den Wert $\delta_m = 0$ an, d.h. das Netz erkennt den Datensatz eindeutig und fehlerfrei als zur Trainingsmenge m bzw. zur Klasse=m gehörend.

Die Trainingsmengen teilt man vorab in m Untermengen gemäß den Klassen auf, wobei jede Trainingsmenge ein eigenes neuronales Hopfield-Netz erhält. Bei der eigentlichen Klassifizierung gewinnt jenes neuronale Netz, das den geringsten Fehlerwert $\delta_m = 0$ bzw. den geringsten Wert für die Integrity (Gl. 3.98) aufweist.

$$r = \left(1 - \frac{\delta_{min}}{\delta_{max}}\right) \cdot 100\% \quad \text{mit} \quad 0 \leq r \leq 100\% \tag{3.98}$$

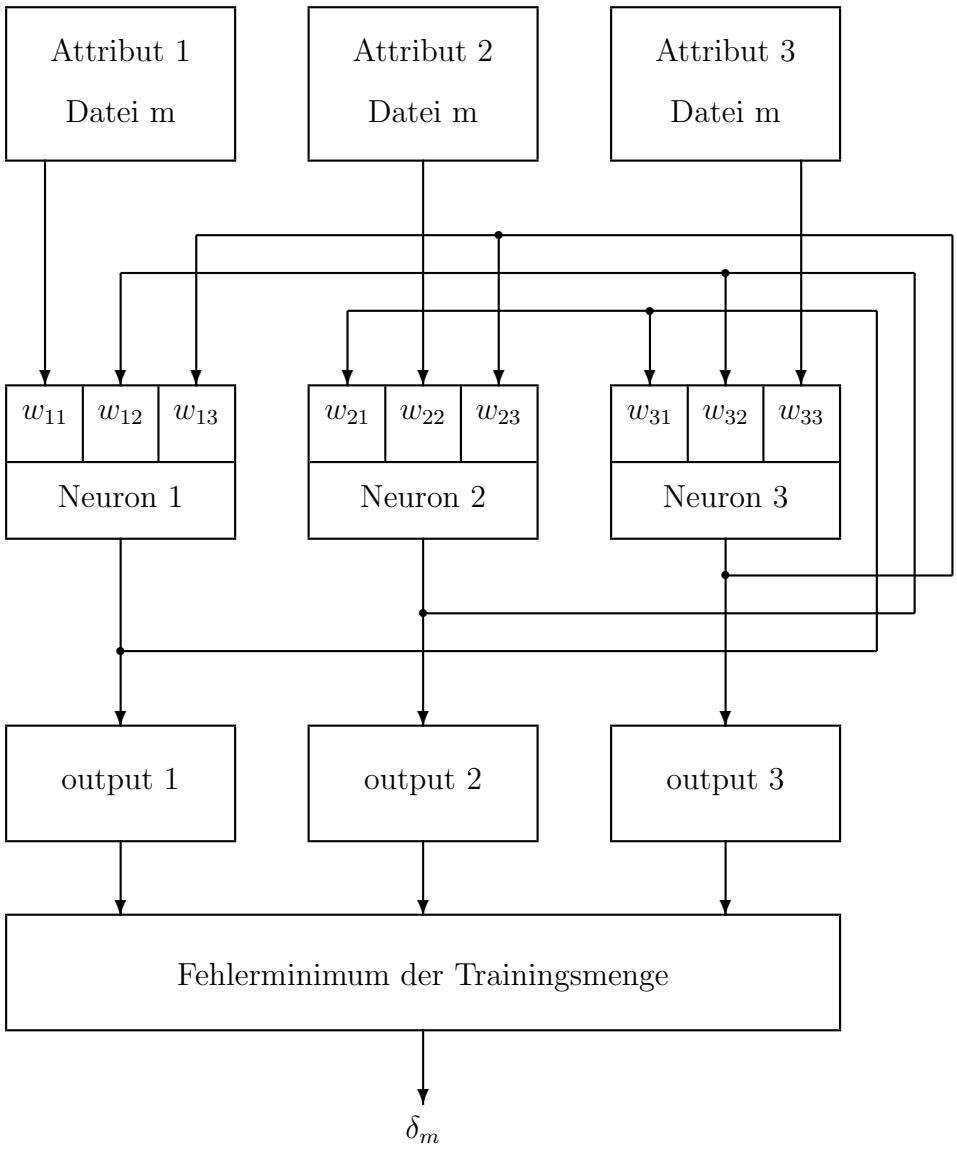


Abb.: 3.16: Neuronales Hopfield-Netz für drei Attribute

3.6.2 Binäre Hopfield-Netze

Hopfield-Netze arbeiten besonders effektiv mit binären Datenformaten, wie z.B. im ± 1 Definitionsbereich. Für die Stufen eines konkreten Datensatzes, die in der Regel nach der Skalierung als ganzzahlige Werte zwischen 0 und einem maximalen Wert x_{max} vorliegen, wählt man eine Konvertierung, die für die Zahl 7 in einer 8 Bit-Darstellung die folgenden „Bits“ ergibt:

$$7 \quad \longrightarrow \quad (0,0,0,0,0,1,1,1) \quad \longrightarrow \quad (-1,-1,-1,-1,-1,1,1,1) \tag{3.99}$$

Durch die Verwendung der negativen Eins entsteht ein numerisch günstiges und für die Hardlimiter-Schaltfunktion des Hopfield-Netzes optimales Format.

Die Anzahl der Neuronen, die eigentlich der Anzahl der Attribute entspricht, erhöht sich durch die binäre Darstellung um den Faktor der maximalen Stellenzahl. In der Abb. 3.16 erhöht sich Anzahl der Neuronen bei einer Byte-Darstellung nun von 3 auf $3 \cdot 8 = 24$. Dadurch fällt mehr Rechen- und Speicheraufwand an, allerdings bieten dadurch auch die folgenden entscheidenden Vorteile an:

- Die Anzahl der Datensätze, die ein Hopfield-Netz speichern kann, steigt näherungsweise mit der der 3. Wurzel der maximalen Anzahl der Bits an. Eine Byte-Darstellung führt z.B. zu einer Verdopplung der Speicherkapazität des Hopfield-Netzes.
- Die Wahl einer günstigen Stellenzahl, oberhalb der für x_{max} erforderlichen Anzahl, ermöglicht eine Optimierung der Anzahl der Neuronen. Dies erweist sich bei der Anpassung des Hopfield-Netzes an eine variable Anzahl von Datensätzen als besonders vorteilhaft.
- Die Verwendung von ± 1-Ziffern unterstützt eine effektive Programmierung mit boolschen Datentypen, denn eine -1 lässt sich als ein logisches „false“ interpretieren. Dadurch kompensieren sich die erhöhten Anforderungen an Rechen- und Speicheraufwand erheblich.

Beispiel: Hopfield-Netz für 10 Datensätze

Als nachvollziehbares Zahlenbeispiel eignet sich die in der Tab.: 3.14 dargestellte Datei, die aus Gründen der Anschaulichkeit nur aus 10 Datensätzen besteht und schon im Abschnitt über die Klassifikation mittels Entscheidungsbaum und im Abschnitt über die Klassifikation mittels Support Vektor Maschinen als Beispiel diente.

Diese Datei ist zu Beginn in zwei Dateien zu zerlegen und zwar entsprechend der Zugehörigkeit eines Trainingsdatensatzes zu einer Klasse. Damit entstehen zwei Datenbestände, die in kompakter Matrizenform wie folgt (3.100)aussehen:

$$A = \begin{pmatrix} 7 & 1 & 5 \\ 6 & 5 & 4 \\ 4 & 4 & 6 \\ 8 & 2 & 7 \\ 3 & 4 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 3 & 3 \\ 2 & 3 & 3 \\ 1 & 1 & 2 \\ 3 & 2 & 3 \\ 3 & 3 & 4 \end{pmatrix} \tag{3.100}$$

Der Wert der numerisch größten Stufe beträgt 8, d.h. für die Konvertierung auf ein binäres Format genügen 4 Bits. Das Fehlen der Stufenwerte 1 und 2 beim ersten Attribut hängt mit der geringen Anzahl der Datensätze zusammen. Programmtechnisch ist es günstig, die Stufenwerte als Index zu implementieren.

Basierend auf einer 4-Bit Darstellung und des ±1 Zahlenbereiches, stellen die Gl.: 3.101 und 3.101 die konvertierten Datensätze dar.

$$A = \begin{pmatrix} -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \end{pmatrix} \tag{3.101}$$

$$B = \begin{pmatrix} -1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \end{pmatrix} \tag{3.102}$$

Mittels der in den Gl. 3.101 und Gl. 3.101 enthaltenen Werte entstehen durch die Auswertung der Gl. 3.96 für die beide Hopfield Netze symmetrische Gewichtsmatrizen 3.103, 3.104, die nur beim Programmstart zu berechnen sind und auf deren Anwendung die eigentliche Klassifizierung basiert. Die Anzahl der Zeilen dieser quadratischen Matrizen entspechnen der mit der Anzahl der Bits multiplizieren Anzahl der Attribute, also $4 \cdot 3 = 12$.

$$\begin{pmatrix} 1.0 & -0.6 & -0.6 & -0.2 & 0.6 & -0.6 & 0.1 & -0.2 & 0.6 & -0.6 & 0.6 & 0.6 \\ -0.6 & 1.0 & 0.2 & -0.2 & -0.2 & 0.2 & -0.6 & 0.6 & -0.2 & 0.2 & -0.2 & -0.2 \\ -0.6 & 0.2 & 1.0 & 0.6 & -0.2 & 0.2 & -0.6 & 0.6 & -0.2 & 0.2 & -0.1 & -0.2 \\ -0.2 & -0.2 & 0.6 & 1.0 & 0.2 & -0.2 & -0.2 & 0.2 & 0.2 & -0.2 & -0.6 & 0.2 \\ 0.6 & -0.2 & -0.2 & 0.2 & 1.0 & -0.2 & 0.6 & 0.2 & 0.1 & -0.1 & 0.2 & 0.2 \\ -0.6 & 0.2 & 0.2 & -0.2 & -0.2 & 1.0 & -0.6 & -0.2 & -0.2 & 0.2 & -0.2 & -0.1 \\ 0.1 & -0.6 & -0.6 & -0.2 & 0.6 & -0.6 & 1.0 & -0.2 & 0.6 & -0.6 & 0.6 & 0.6 \\ -0.2 & 0.6 & 0.6 & 0.2 & 0.2 & -0.2 & -0.2 & 1.0 & 0.2 & -0.2 & -0.6 & 0.2 \\ 0.6 & -0.2 & -0.2 & 0.2 & 0.1 & -0.2 & 0.6 & 0.2 & 1.0 & -0.1 & 0.2 & 0.2 \\ -0.6 & 0.2 & 0.2 & -0.2 & -0.1 & 0.2 & -0.6 & -0.2 & -0.1 & 1.0 & -0.2 & -0.2 \\ 0.6 & -0.2 & -0.1 & -0.6 & 0.2 & -0.2 & 0.6 & -0.6 & 0.2 & -0.2 & 1.0 & 0.2 \\ 0.6 & -0.2 & -0.2 & 0.2 & 0.2 & -0.1 & 0.6 & 0.2 & 0.2 & -0.2 & 0.2 & 1.0 \end{pmatrix} \tag{3.103}$$

$$\begin{pmatrix} 1.0 & 0.1 & -0.2 & -0.6 & 0.1 & 0.1 & -0.6 & -0.6 & 0.1 & 0.6 & -0.6 & -0.2 \\ 0.1 & 1.0 & -0.2 & -0.6 & 0.1 & 0.1 & -0.6 & -0.6 & 0.1 & 0.6 & -0.6 & -0.2 \\ -0.2 & -0.2 & 1.0 & -0.2 & -0.2 & -0.2 & 0.6 & -0.2 & -0.2 & 0.2 & -0.2 & 0.2 \\ -0.6 & -0.6 & -0.2 & 1.0 & -0.6 & -0.6 & 0.2 & 0.2 & -0.6 & -0.2 & 0.2 & -0.2 \\ 0.1 & 0.1 & -0.2 & -0.6 & 1.0 & 0.1 & -0.6 & -0.6 & 0.1 & 0.6 & -0.6 & -0.2 \\ 0.1 & 0.1 & -0.2 & -0.6 & 0.1 & 1.0 & -0.6 & -0.6 & 0.1 & 0.6 & -0.6 & -0.2 \\ -0.6 & -0.6 & 0.6 & 0.2 & -0.6 & -0.6 & 1.0 & 0.2 & -0.6 & -0.2 & 0.2 & 0.6 \\ -0.6 & -0.6 & -0.2 & 0.2 & -0.6 & -0.6 & 0.2 & 1.0 & -0.6 & -0.2 & 0.2 & -0.2 \\ 0.1 & 0.1 & -0.2 & -0.6 & 0.1 & 0.1 & -0.6 & -0.6 & 1.0 & 0.6 & -0.6 & -0.2 \\ 0.6 & 0.6 & 0.2 & -0.2 & 0.6 & 0.6 & -0.2 & -0.2 & 0.6 & 1.0 & -0.1 & -0.6 \\ -0.6 & -0.6 & -0.2 & 0.2 & -0.6 & -0.6 & 0.2 & 0.2 & -0.6 & -0.1 & 1.0 & 0.6 \\ -0.2 & -0.2 & 0.2 & -0.2 & -0.2 & -0.2 & 0.6 & -0.2 & -0.2 & -0.6 & 0.6 & 1.0 \end{pmatrix} \tag{3.104}$$

Zum Testen des Hopfield-Netzes legt man die bereits trainierten Datensätze ein zweites Mal an. Für diesen Fall muß immer eines der Netze beim Vergleich der Ausgangssignale mit dem angelegten Datensatz einen Fehler=0 ergeben. Das Netz mit der anderen Klassen liefert dagegen einen von Null verschiedenen Fehlerwert zurück. Das Netz mit dem geringeren Fehler ordnet schließlich die gesuchte Klasse zu.

Die Tab.: 3.19 zeigt, wie erwartet, die korrekten Ergebnisse, d.h. alle 10 Datensätze wurden richtig klassifiziert. Der Fehlerwert von z.B. 5 bedeutet, dass genau 5 Bits vom richtigen Bitmuster abweichen.

Nr.	Fehler Netz 1	Fehler Netz 2	erkannte Klasse	Referenzwert der Klasse
1	0	3	10	10
2	0	4	10	10
3	4	0	1	1
4	0	5	10	10
5	0	4	10	10
6	4	0	1	1
7	5	0	1	1
8	4	0	1	1
9	3	0	1	1
10	0	3	10	10

Tab.: 3.19: Ergebnisse des Hopfield-Netzes für 10 Datensätze

Für konkrete Anwendungen sollten sich die Fehlerwerte der nicht zutreffenden Klassen möglichst deutlich vom Fehlerwert der zutreffenden Klasse unterscheiden. Der relative Fehlerwert eignet sich daher als Integritywert. Bei einer Byte-Darstellung der Attributswerte kann der maximale Fehler 8x2, d.h. 16 Einheiten betragen. Damit folgt gemäß der Formel für die Integrity bei einem Fehler 0 ein Wert von 100%. Beim 4. Datensatz liefert die Gl. 3.105 einen Wert von

$$r = \left(1 - \frac{5}{16}\right) = 68,75\% \tag{3.105}$$

Dieser Wert für eine falsche Klasse unterscheidet sich deutlich von den 100% für die richtige Klasse.

Bei der programmtechnischen Umsetzung eines Hopfield-Netzes sind die folgenden Einsparungen an Rechenzeit und Speicherplatz möglich:

- Wegen der Symmetrie der Gewichte genügt es, nur die eine Hälfte der Matrix 3.103 bzw. der Matrix 3.104 zu berechnen.
- Die Division durch die Anzahl M der Datensätze in der Summenformel 3.96 braucht nicht ausgeführt zu werden, wenn dafür das Gewicht der Eingangskanäle auf M gesetzt wird. Dies bedeutet eine enorme Reduktion an Rechenzeit und Speicherplatz, denn die alle Berechnungen für das Hopfield-Netz lassen dann ausschließlich im Integer-Format ausführen.
- Das Gewicht der Eingangskanäle, das wie eben begründet, auf den Wert M gesetzt wird, ermöglicht durch eine geringe prozentuale Abweichung von M eine Feineinstellung des Hopfield-Netzes.
- Die Auswertung des Hopfield-Netzes bei der eigentlichen Klassifizierung verkürzt sich durch eine Abfrage auf eine Änderung der Ausgangssignale zum vorherigen Rechentakt. Meistens reichen ca. 10 Iterationen aus, die sich jedoch bei unveränderlichen Ausgangssignalen abbrechen lassen.

3.6.3 Ablaufsteuerung eines Hopfield-Netzes

Die Abb. 3.17 veranschaulicht die Ablaufsteuerung eines Hopfield-Netzes zur Klassifizierung eines Datenbestandes in drei Klassen. drei Klassen.

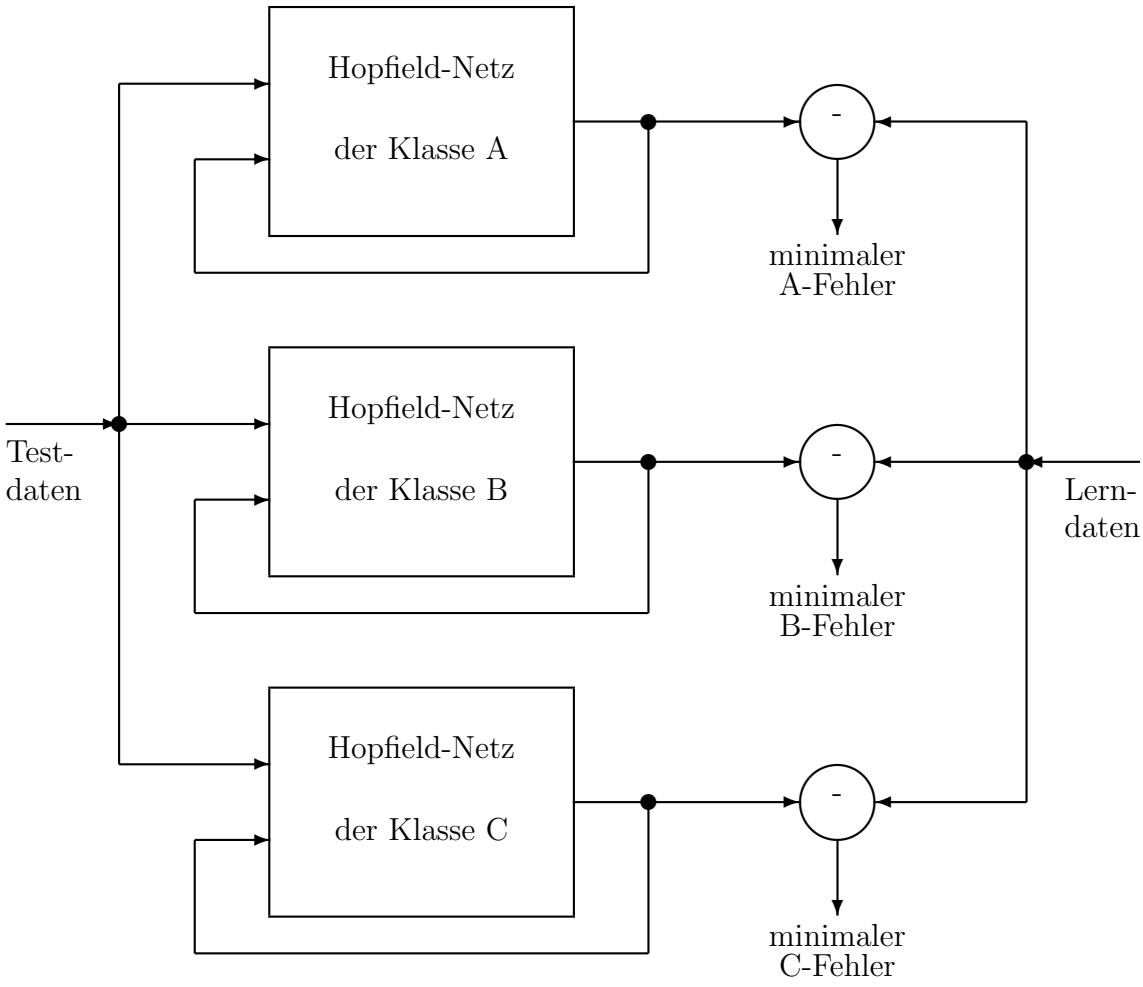


Abb.: 3.17: Ablaufsteuerung bei neuronalen Hopfield-Netzen

Die Testdaten stellen die zu klassifizierenden Datensätze dar. Jedes der drei unabhängigen Netze liefert einen Ausgangsdatensatz.

Die Lerndaten, deren Datensätze die Klassen enthalten, spaltet man in drei Teile gemäß den drei Klassen auf und verwendet diese Teildatenbestände zur Berechnung der jeweiligen Gewichtsmatrizen der drei Hopfield-Netze A,B und C während der Initialisierungphase.

Die drei Ausgangsdatensätze aus den Netzen A, B und C werden mit den jeweiligen Lerndaten der Teildatenbestände verglichen und unabhängig für jede Klasse der kleinste Betrag der entsprechenden Vektordifferenz berechnet.

Das Minimum aus den jeweils kleinsten Beträgen der Netze A, B und C entspricht dann der Klasse des Testdatensatzes.

Kapitel 4

Clusterbildung

Die Verfahren der Clusterbildung unterscheidet sich von den Klassifikationsverfahren dadurch, dass die Zuordnung der Trainingsdaten zu einer Klasse grundsätzlich unbekannt ist. Liegt z.B. ein Datensatz in Form einer Bildersammlung vor, dann entspricht die Clusterbildung dem Sortieren dieser Sammlung nach den unterschiedlichen Bildinhalten. Zur sprachlichen Unterscheidung bezeichnet man die Clusterbildung auch als Gruppierung. Ein Algorithmus zur Clusterbildung sortiert die Datensätze in Gruppen mit gemeinsamen Eigenschaften, die sich von den Eigenschaften der anderen Gruppen deutlich unterscheiden. Im anschaulichen Beispiel einer Bildersammlung könnten sich die Bilder nach den jeweils in ihnen enthaltenen Farbanteilen grupppieren.

Clusterbildung gehört zum Lernprozeß beim Menschen. Ein Kind gruppiert („clustert“) Haustiere z.B. in Hunde und Katzen, obwohl es nicht in der Lage ist, eine genaue Definition zu entwickeln, was Hund und Katze unterscheidet. Man bezeichnet diese Art des Lernen als „unüberwachtes Lernen“, denn das Kind besitzt keinen Kriterienkatalag nach dem es die Haustiere unterscheidet.

Das wirtschaftliche Interesse an der Clusterbildung besteht darin, spezifische Kundengruppen in einem Datenbestand zu erkennen, ohne dass dafür Auswahlkriterien zur Verfügung stehen. Solche Kundengruppen fallen durch ein von der Mehrheit der anderen Datensätze abweichendes Datenprofil auf, das z.B. Geldwaschaktivitäten oder Versicherungsbetrug vermuten lässt.

Die meisten Verfahren zur Clusterbildung basieren auf einer Abstandsberechnung der Datensätze. Stellt man sich z.B. Datensätze mit drei Attributen vor, dann ergibt die graphische Darstellung im 3D-Vektorraum mehrere Punktwolken, wobei jede zusammenhängende Punktwolke einer Gruppe, d.h. einem Cluster, entspricht. Der Cluster-Algorithmus lernt durch Beobachten und nicht, wie beim Klassifizieren, durch die bekannten Beispielklassen eines Trainingsdatensatzes.

Die Probleme, die bei der Clusterbildung auftreten können, sind die gleichen wie beim Klassifizieren. Vor allem die Skalierung der Rohdaten bei unterschiedlichen Einheiten des Attribute beeinflusst ganz wesentlich das Ergebnis der Clusterbildung. Daher ist es besonders wichtig, die Datensätze in geeigneter Weise zu normieren und von Fehlern und Widersprüchen möglichst weitgehend zu bereinigen.

Neben dem bereits erwähnten Abstandsverfahren bietet sich die spezielle neuronale Netzstruktur einer Kohonen-Map für die Clusterbildung an. Die entsprechende Feature Map visualisiert den Fortschritt der rekursiven Clusterbildung besonders anschaulich und das Kohonen-Netz ist in seiner Leistungsfähigkeit allen anderen Clusterbildungsverfahren weit überlegen. Allerdings erfordert das Trainieren einer Feature Map beim Programmstart einen erheblichen Rechenaufwand.

4.1 Abstandsverfahren

Abstandsverfahren basieren auf der Berechnung von Abständen zwischen einer geringen Anzahl von ausgewählten Datensätzen, die sich als Mittelpunkte der Cluster interpretieren lassen und allen anderen Datensätzen. Interpretiert man die Zahlenwerte der m Attribute eines Datensatzes als die Kompententen eines Vektors $x = \{x_1, \ldots, x_m\}$, dann entspricht die Länge des Differenzvektors zwischen einem Datensatz i und einem Datensatz j dem gesuchten Abstand, der sich allgemein wie folgt berechnen lässt:

$$d(i,j) = \sqrt{w_1|x_{i,1} - x_{j,1}|^p + \cdots + w_n|x_{i,m} - x_{j,m}|^p}; \qquad \sum_{i=1}^{m} w_i = 1; \tag{4.1}$$

Das Addieren der Beträge für p=1 bietet bei der Clusterbildung den Vorteil, dass Ausreißer weniger stark in die Abstandsberechnung eingehen. Mit der Gewichtung w lässt sich die Wahrscheinlichkeit oder die Bedeutung der einzelnen Atrribute modellieren.

Eine besondere Bedeutung kommt der einheitlichen Normierung der Attributswerte zu. Damit eine große Differenz zwischen einem Datensatzpaar eine entsprechende Aussage erlaubt, müssen alle Zahlenwerte in einem einheitlichen Definitionsbereich liegen, wie z.B. in einem Bereich von ± 100, was eine speichersparende Byte-Darstellung ermöglicht.

Beispiel mit RGB-Farben und 10 Datensätzen

Das Verhalten einer typischen Clusterbildung lässt sich gut mit Farbdarstellungen veranschaulichen. Die Tab.: 4.1 zeigt links eine Anzahl von 10 zufällig gewählten und auf ±127 skalierten RGB-Farbkombinationen. Die rechts dargestellte Tabelle enthält die entsprechenden Abstände gemäß der Gl. 4.2, welche einer symmetrischen Abstandsmatrix entspricht.

$$d(i,j) = \frac{0.5}{3} * (|x_{i,1} - x_{j,1}| + |x_{i,2} - x_{j,2}| + |x_{i,3} - x_{j,3})|) \tag{4.2}$$

Der Faktor 0.5 begrenzt die Abstände auf den Definitionsbereich des Java Byte-Typs.

n	rot	grün	blau
0	-3	45	41
1	5	4	-17
2	73	-118	14
3	29	-82	-30
4	-80	40	40
5	108	105	-40
6	84	-13	106
7	68	52	-66
8	12	96	-125
9	-85	-39	47

⟶

	0	1	2	3	4	5	6	7	8	9
0	0	17	44	38	13	42	35	30	38	28
1	17	0	36	20	29	37	36	26	34	32
2	44	36	0	20	56	52	34	42	69	45
3	38	20	20	0	50	46	43	34	48	39
4	13	29	56	50	0	55	47	44	52	15
5	42	37	52	46	55	0	48	19	31	70
6	35	36	34	43	47	48	0	42	68	42
7	30	26	42	34	44	19	42	0	26	59
8	38	34	69	48	52	31	68	26	0	67
9	28	32	45	39	15	70	42	59	67	0

Tab.: 4.1: Zufällige Farbpunkte und Abstandsmatrix

Eine weit verbreitete Methode der Clusterbildung besteht nun darin, eine Auswahl von n Punkten als die n Mittelpunkte der n Cluster anzunehmen und die restlichen Punkte gemäß ihrem minimalen Abstand diesen Mittelpunkten zuzuordnen. Die Summe aller Abstände für jedes Cluster ist dann für alle Kombinationen der möglichen Mittelpunkte zu minimieren.

Für das in der Tab.: 4.1 dargestellte Beispiel könnte man mit den Punkten 3 und 6 beginnen. Dazu liefert die Abstandsmatrix die entsprechenden Zuordnungen für die verbleibenden 8 Punkte. Beginnend mit dem Punkt 0 folgt ein Abstand von 38 zum Mittelpunkt 3 und ein Abstand

von 35 zum Mittelpunkt 6. Somit gehört wegen des geringeren Abstandes der erste Punkt 0 zum Cluster, das sich um den Mittelpunkt 6 gruppiert. Diese Zuordnungen führt man für alle 8 Punkte durch und segmentiert somit die Punktmenge in zwei Cluster.

Im nächsten Schritt ist nun die Summe aller Abstände der Punkte vom Mittelpunkt ihres Clusters zu berechnen. Die weiteren Abstände zeigt die Tab.: 4.2:

n	Mittelpunkt 3	Mittelpunkt 6
0	(38)	35
1	20	(36)
2	20	(34)
3	0	(43)
4	(50)	47
5	46	(48)
6	(43)	0
7	34	(42)
8	27	(68)
9	(48)	42
Summe	186	124

Tab.: 4.2: Abstandswerte

Die Summe aller nach der Clusterbildung verbleibenden Abstände beträgt somit 186+124 = 310. Der iterative Algorithmus wählt nun eine neue Kombination von Mittelpunkten und wiederholt die Berechnungen solange, bis die Summe der verbleibenden Abstände ein Minimum annimmt. Falls diese Summe im nächsten Schritt mit neuen Mittelpunkten jedoch ansteigt, verwirft man diese Iteration und speichert nur bei einer Verkleinerung der Summe die Cluster ab.

Für den Fall von drei angenommenen Mittelpunkten entstehen drei Cluster für die verbleibenden 7 Punkte. Das Clusterverfahren ist beendet, wenn sich bei einer bestimmten Anzahl von Mittelpunkten das Minimum der Abstandssumme ergibt.

Beispiel mit RGB-Farben und 100 Datensätzen

Die Abb. 4.1 visualisiert in Form von Farbbalken das Ergebnis des beschriebenen Abstandsverfahrens für zufällig gewählte 100 Farben, die um 6 Mittelpunkte gruppiert wurden. Die oberste Reihe der Farbbalken stellt die gegebenen Farben dar. Der jeweils erste Farbbalken am Beginn der nachfolgenden Cluster zeigt die Farbe des nach 1000 Iterationen ermittelten Mittelpunkte. Deutlich ist die Nähe der nach rechts aufgetragenen einzelnen Farbbalken zum jeweiligen Mittelpunkt am Anfang der Clusterstreifen zu erkennen.

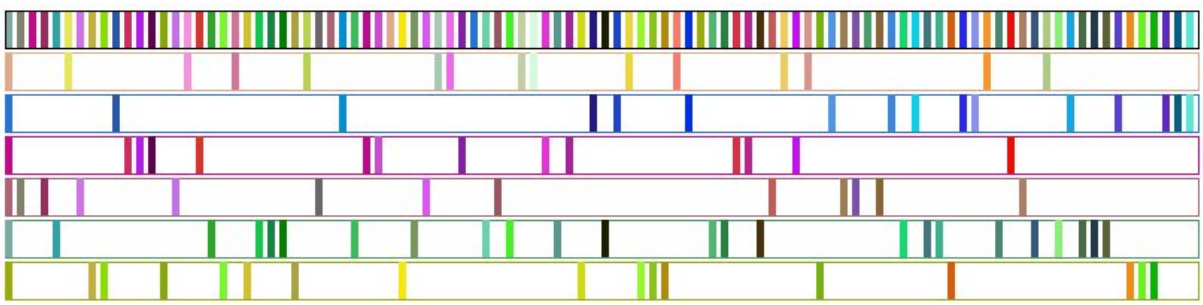


Abb.: 4.1: Gruppierung von Farbbalken in 6 Cluster

Die Anzahl von nur 6 Mittelpunkten ist offensichtlich zu gering, um die Vielzahl der unterschiedlichen Farben zu gruppieren. Daher tauchen z.B. in der untersten grünen Farbbalkenreihe falsch gruppierte hellbraune Farbbalken auf.

Zum Vergleich zeigt die Abb. 4.2 die Gruppierungen für 16 Mittelpunkte bei ebenfalls zufällig gewählten 100 Farben. Wie zu erkennen ist, enthalten die einzelnen Cluster nun die jeweils zusammengehörenden Farben. Je höher man die Anzahl der Mittelpunkt wählt, desto besser gelingt die Aufteilung der Farben in Gruppen. Eine fehlerfreie Gruppierung stellt sich dann ein, wenn die Zahl der Mittelpunkte mit der Anzahl der unterschiedlichen Farben übereinstimmt. Allerdings liegt dann keine Gruppenbildung mehr vor.

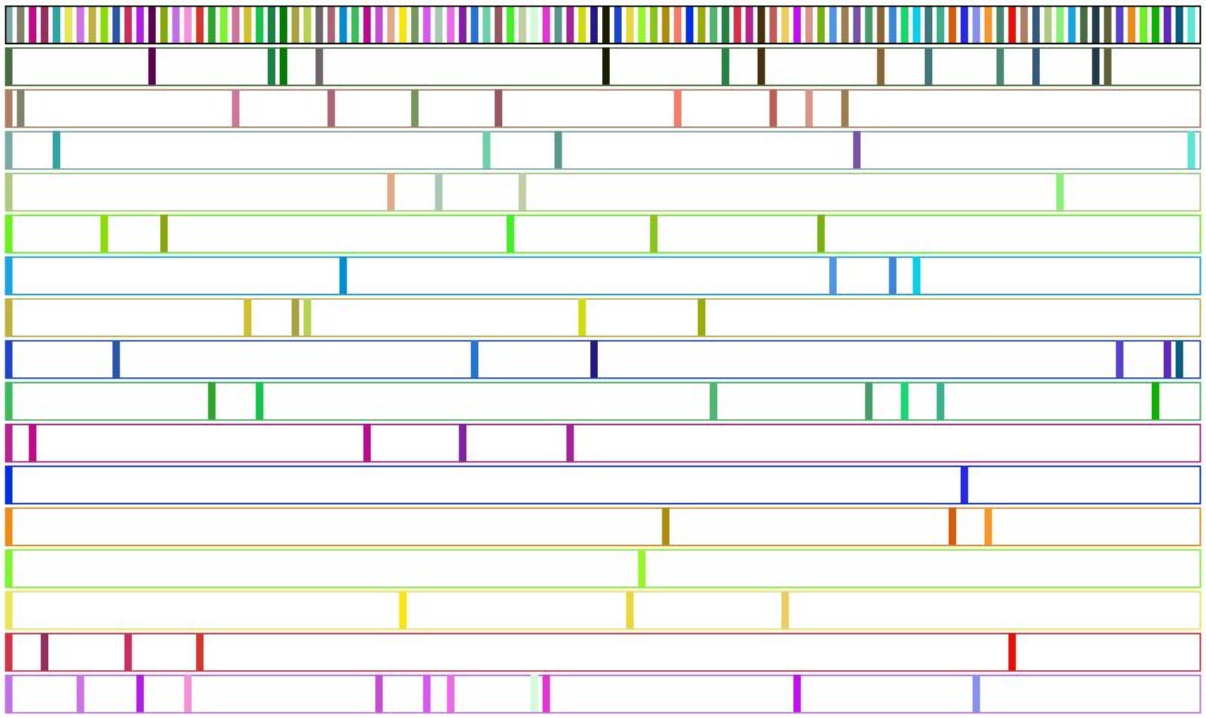


Abb.: 4.2: Gruppierung von Farbbalken in 16 Cluster

Ein interessanter Sonderfall ergibt sich bei 3 Mittelpunkten, wenn die die gegebenen 100 Farben nur aus den drei Grundfarben RGB bestehen. Jetzt gruppieren sich die Cluster fehlerfrei in die RGB-Farben.

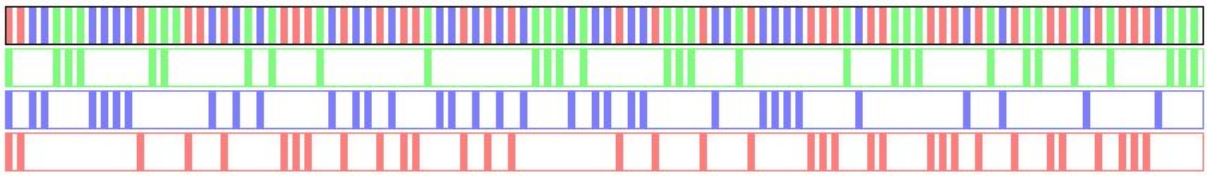


Abb.: 4.3: Gruppierung von RGB-Farbbalken in 3 Cluster

Bewertung der Abstandsverfahren

Abstandsverfahren erfordern eine manuelle und in diskreten Stufen vom Anwender vorzugebende Anzahl der Cluster, was einen gravierenden Nachteil bedeutet. Die Konvergenz der Iteration ist nicht grundsätzlich sichergestellt, lässt sich jedoch durch verfeinerte Optimierungsalgorithmen, wie z.B. einem genetischen Algorithmus, verbessern. Der besondere Vorteil der Abstandsverfahren liegt in der einfachen Programmierung. Allerdings steigt bei einer großen Anzahl von Datensätzen mit vielen möglichen Mittelpunkten der Rechenaufwand enorm an.

Ausreißer lassen sich bei Abstandsverfahren gut erkennen, denn ihr Abstand zu ihrem Cluster-Mittelpunkt ist statistisch gesehen erheblich größer als der mittlere Abstand eines Datensatzes zu seinem Cluster.

4.2 Neuronale Kohonen-Netze

Stellt man sich ein neuronales Netz mit beliebigen Verbindungen vor, dessen Neuronen jeweils eine Leuchtdiode tragen, dann lässt sich die Aktivität der Neuronen durch ein Aufleuchten seiner Diode visualisieren. Die Abb. 4.4 zeigt in der linken oberen Hälfte den Zustand des Netzes nach dem Anlegen eines Eingangsmusters A an. Die Darstellung oben rechts visualisiert den Zustand dieses neuronalen Netzes nach dem Anlegen eines Eingangsmusters B. Das Aufleuchten der Dioden hängt vom Trainingszustand des neuronalen Netzes ab und die aktiven Neuronen weisen keine Ordnungsmuster auf.

Die unteren beiden Bilder der Abb.: 4.4 stellen die gleiche Situation beim Anlegen des Musters A (Bild links unten) und des Musters B (Bild rechts unten) dar, allerdings sorgte bei diesem neuronalen Netz der Trainingsalgorithmus dafür, dass die Art des angelegten Musters aus der Position der aktiven Neuron ablesbar ist. Immer wenn ein dem Muster A ähnliches Muster am neuronalen Netz anliegt, aktivieren sich die Neuronen in der oberen linken Ecke und dem Muster B ähnliches Eingangssgnale erkennt man am Aufleuchten der Dioden in der rechten unteren Ecke. Man bezeichnet eine derartige Netzstruktur, die von T. Kohonen [21] erstmalig veröffentlicht wurde, als eine selbstorganisierende Karte (SOM) und die entsprechenden graphischen Darstellungen als Feature Map.

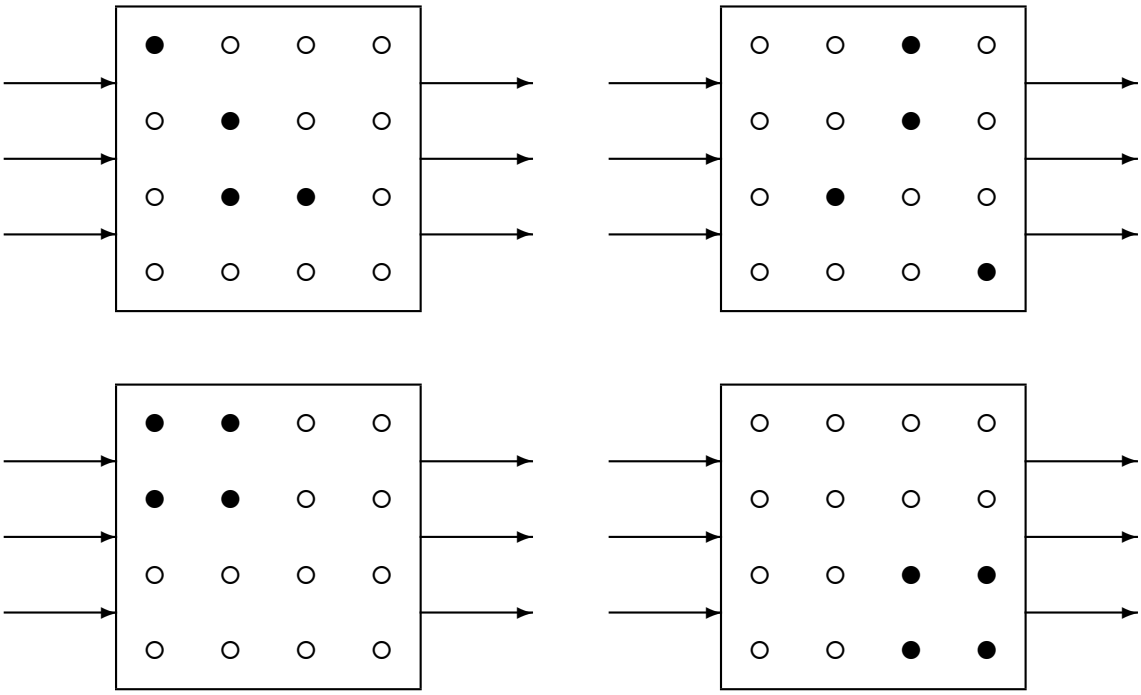


Abb.: 4.4: Zufälliges und selbstorganisierendes Neuronen-Cluster

Selbstorganisierende Karten empfinden die Aktivität der natürlichen Neuronen im biologischen Gehirn nach, denn die Messung der Gehirnströme lässt auch hier abgegrenzte Bereiche erkennen, die bei bestimmten Gehirnfunktionen „aufleuchten“, wie dies aus der Abb.: 4.5 hervor geht. Ein künstliches neuronales Netz sollte ähnliche Leuchtmuster zeigen, denn in diesem Fall sind die Schaltwege minimiert und es entsteht eine rechnende Intelligenz, die sich hervorragend zur maschinellen Clusterbildung eignet.

Die Abb. 4.6 stellt den Aufbau des von T. Kohonen vorgeschlagenen neuronalen Netzes dar. Die Eingangssignale gelangen parallel auf alle Neuronen. Für jedes Neuron ist das Skalarprodukt aus dem Vektor der Eingangssignale und dem Vektor der neuronalen Gewichte zu bilden.

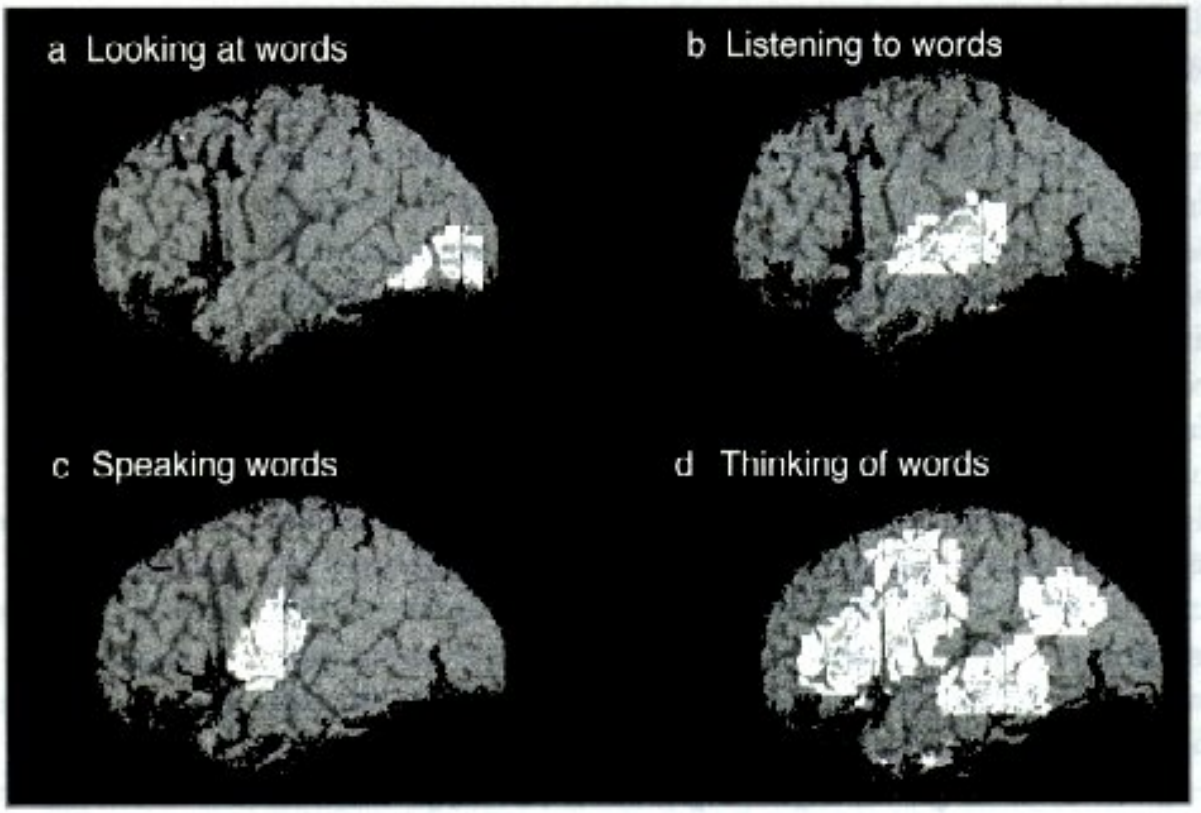


Abb.: 4.5: Aktive Gehirnbereiche (Spektrum der Wissenschaft, Dossier 4/97)

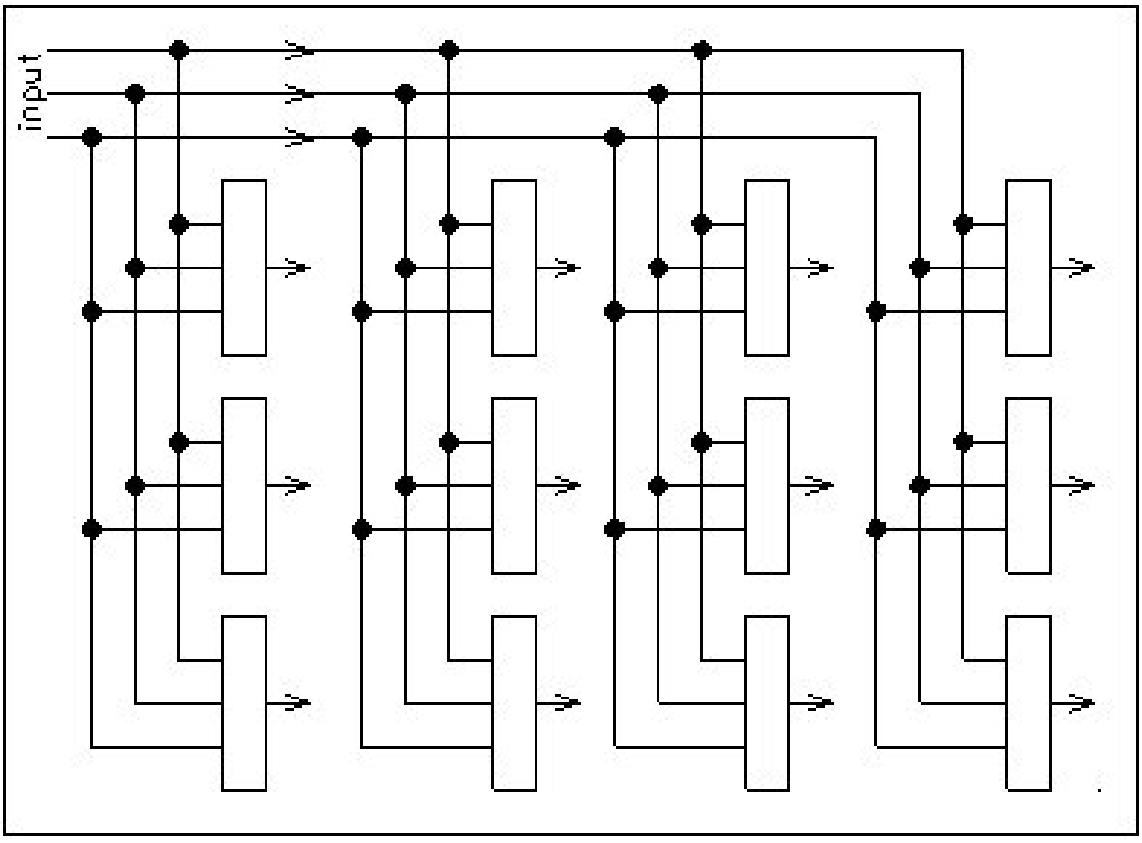


Abb.: 4.6: Aufbau eines neuronalen Kohonen-Netzes

Das Lernprinzip ist denkbar einfach:

Das Neuron mit dem größten Betrag des Skalarproduktes „gewinnt“.

Das gewinnende Neuorn beeinflusst mit seinen Gewichten die Gewichte der umliegenden Neuronen, die zu Beginn der Trainingsläufe zufällige Werte annehmen. An dieser Stelle sei nochmal auf eine sorgfältige Normierung der Eingangssignale hingewiesen, denn sonst würde durch ein zu groß skaliertes Eingangssignal das falsche Neuron gewinnen.

Für die Programmierung empfehlen sich vektororientierte Schreibweisen. Numerisch ist es günstiger, statt dem Betrag des Skalarproduktes die Länge des Differenzvektors aus dem Vektor der Eingangssignale und dem Vektor der Neuronengewichte zu berechnen, wobei dann das Neuron mit dem minimalen Betrag dieses Differenzvektors „gewinnt“.

$$
\begin{array}{ll}
\text{Vektor der Eingangssignale:} & x = \{x_1, \ldots, x_k\} \\
\text{Matrix der Neuronengewichte:} & W = \{w_{i,j,k}\} \\
\text{Zeilen der Feature Map:} & N; \quad 1 \leq i \leq N \\
\text{Spalten der Feature Map:} & M; \quad 1 \leq j \leq M \\
\text{Anzahl der Eingangssignale:} & K; \quad 1 \leq k \leq K \\
\text{Gewinner-Neuron:} & z = \{imin, jmin\} \\
\text{Neuronendifferenz:} & d_{i,j} = \sum_{k=1}^{K} (W_{i,j,k} - x_k)^2 \\
\\
\text{Update des Gewinner-Neurons:} & W^{neu}_{imin,jmin,k} = W^{alt}_{imin,jmin,k} - \gamma \cdot d_{imin,jmin} \\
\\
\text{Update der Umgebung:} & W^{neu}_{i,j,k} = W^{alt}_{i,j,k} - \alpha \cdot \gamma \cdot d_{i,j}; \quad i,j \neq imin, jmin \\
\\
\text{Lernrate für das Gewinner-Neuron:} & \gamma \leq 1.0 \\
\text{Lernrate für die Umgebung:} & \alpha \leq \gamma
\end{array}
\tag{4.3}
$$

Der Algorithmus basiert auf den folgenden Stufen:

1. Zufälliges Setzen der Anfangwerte der 3-dimensionalen Gewichts-Matrix
2. Anlegen des Input-Vektors x und Berechnen aller Differenzen
3. Auswahl des Neurons mit der geringsten Differenz
4. Update der Gewichte in der Umgebung des gewinnenden Neurons
5. Fortsetzung mit Punkt 2 bis die Feature Map stabil bleibt.

Bei speziellen Anwendungen empfielt es sich, die Anfangswerte der Feature Map nicht zufällig zu setzen, sondern mit einigen typischen Datensätzen aus dem Trainingsdaten die Feature Map so zu initialisieren, dass eine möglichst rasche Konvergenz eintritt.

Eine Erweiterung der 2D-Maps stellen die räumlichen Feature Maps dar, bei der auch in der dritten Dimension Neuronen angeordnet werden. Dadurch erhöht sich die maschinelle Intelligenz und gleichzeitig besteht die Möglichkeit einer anschaulichen Visualisierung der Dynamik des Trainingsverlaufes.

Beispiel für die Clusterbildung von 100 RBG-Farben

Die Abb. 4.7 stellt die Feature Map dar, wie sie sich nach 10 Durchläufen bei einem Datensatz von 100 monochromen RGB-Farben ergibt. Die Feature Map besteht aus 8 Zeilen und 12 Spalten. Der Radius der Umgebung eines Gewinner-Neurons liegt bei 3 Neuronen. Die Wert der Lernparmater lauten $\gamma = \alpha = 1.0$. Das neuronale Netz gruppiert die rechts in der Graphiken dargestellten 100 Farben problemlos in die drei RGB-Komponenten.

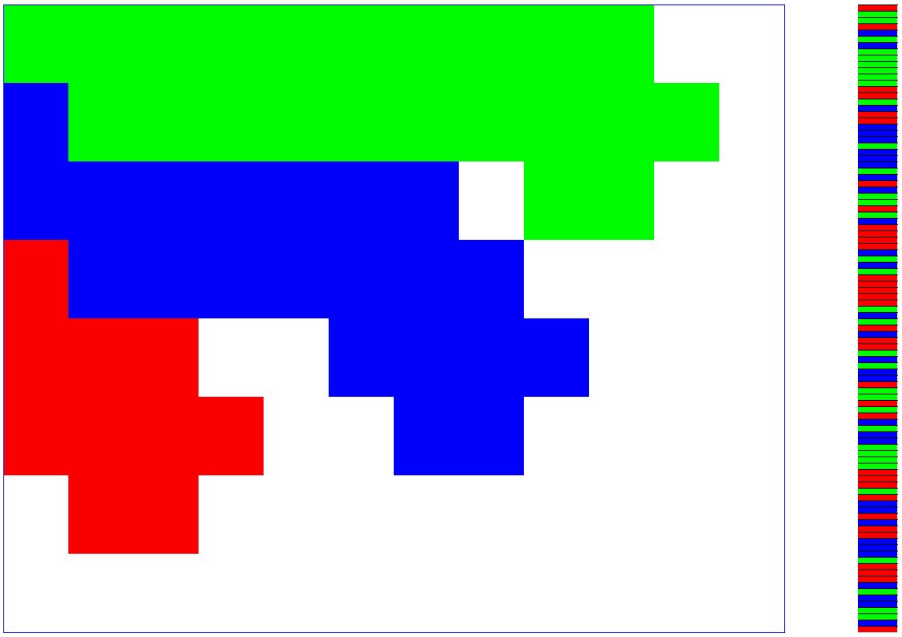


Abb.: 4.7: Feature Map einer Farben-Gruppierung von 100 Farben Rot, Grün oder Blau

Erhöht man die Feature Map auf 40 Zeilen und 50 Spalten und wählt zufällig 100 RGB-Farben aus den ca. 16 Millionen möglichen Farben aus, dann nimmt die Feature Map die in der Abb.: 4.8 gezeigte Gestalt an.

Abb.: 4.8: Feature Map einer Farben-Gruppierung von 100 zufälligen RGB-Farben

Bei einer weiteren Vergrößerung des Neuronegitters auf 110x80 Neuronen und bei einer zu clusternden Serie von 800 RGB-Farben sieht die Feature Map nach 100 Trainingszyklen bzw. nach 30s Rechenzeit die in der Abb.: 4.9 gezeigte Gestalt an. Ein Blick auf die Abb.: 4.9 bestätigt, dass die zufällig gewählten 800 verschiedenen Farbkombinationen nur aus drei RGB Grundfarben bestehen. Dies ist eine erstaunliche Leistung des neuronalen Netzes, das mittels seiner maschinellen Intelligenz autonom diese RGB-Cluster erkennen konnte.

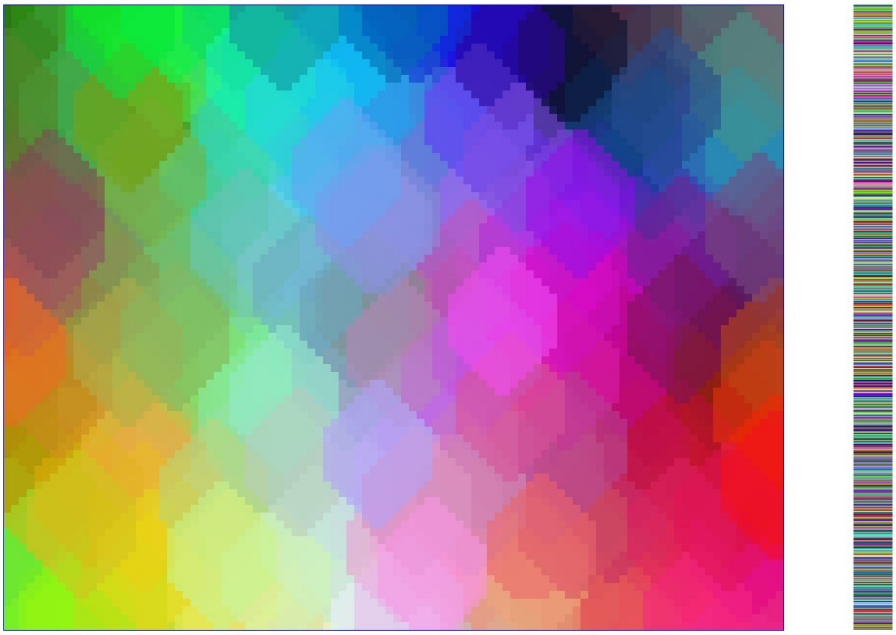


Abb.: 4.9: 120x80 Feature Map für 800 zufällige RGB-Farben

Bei konkreten Anwendungen bietet sich die Kombination der beiden Verfahren des Klassifizierens und des Clusterns an. Dazu clustert man die Trainingsdaten und markiert dann farblich in der Feature Map die bekannten Klassen der Datensätze.

Die Abb.: 4.10 stellt die Feature Map eines neuronalen Kohonen-Netzes dar. Die roten bzw. grünen Bereiche visualisieren die beiden Klassen einer Datei, die aus 8000 protokollierten Datensätzen einer speziellen EBay-Auktion besteht. Jede einzelne Auktion entspricht einem Element in der Graphik. Die schwarzen Bereiche markieren jene Datensätze, die der noch nicht abgeschlossene Trainingsalgorithmus einer falschen Klasse zuordnet. Die Datei stammt aus dem Data Mining Cup 2006 und dient als Datenbasis zur Vorhersage erfolgreicher EBay-Auktionen.

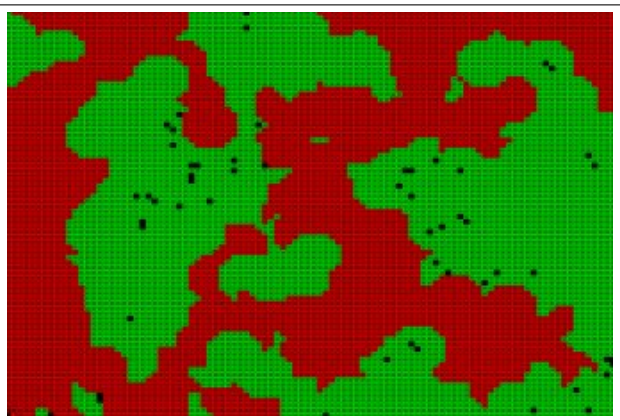


Abb.: 4.10: Kombination von Klassifizieren und Clustern

4.3 Clusterbildung mit graphischen Daten

Das neuronale Kohonen Netz bietet sich für die Clusterbildung von Bildersammlungen an. Bestimmte Teilmengen einer Bildersammlung unterscheiden sich von anderen Teilmengen. Als Beispiel für die Clusterbildung dient eine Fotoserie mit 32 Bildern, die eine Wanderung in den Alpen dokumentieren. Die Bildersammlung teilt man nun in 16 Trainingsbilder und in 16 Testbilder auf. Die Abb.: 4.11 stellt in den beiden oberen Reihen das Trainingsset und in den unteren beiden Bilderreihen das Testset dar.

Abb.: 4.11: Trainings- und Testset einer Bildersammlung

Die Abb.: 4.12 zeigt die Feature Map nach der Initialisierung.

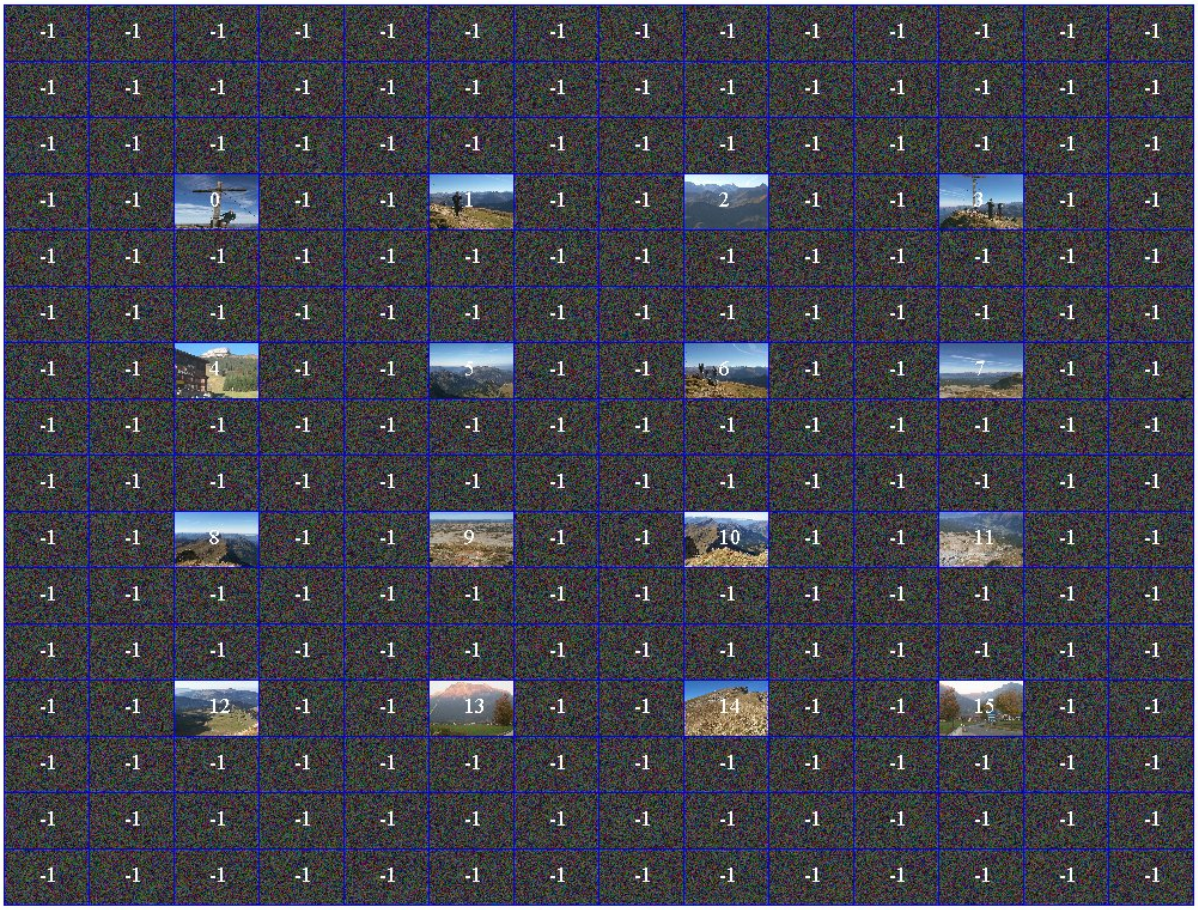


Abb.: 4.12: Anfangszustand der Feature Map nach dem Initialisieren mit dem Trainingsset

Das Setzen der Anfangswerte der Feature Map mit den 16 Bildern des Trainingssets begünstigt die Konvergenz, denn es dauert lange bis aus Zufallszahlen Bilder entstehen, die mit den Trainingsbildern vergleichbar sind. Die einzelnen Elemente der Feature Map tragen die Nummer des jeweiligen Bildes. Die -1 bedeutet, dass Zufallszahlen vorliegen.

Nach 15 Rechenschleifen entsteht die stabile Feature Map der Abb.: 4.13. Deutlich lassen sich die verschiedenen Cluster unterscheiden. Jede Bildgruppe kennzeichnet ein gemeinsames Bildmotiv, wie z.B. die Gipfelbilder im Cluster 3, der Bergrücken des „Hohen Ifen“ im Cluster 14, die Gesteinsformationen im Cluster 11, usw.

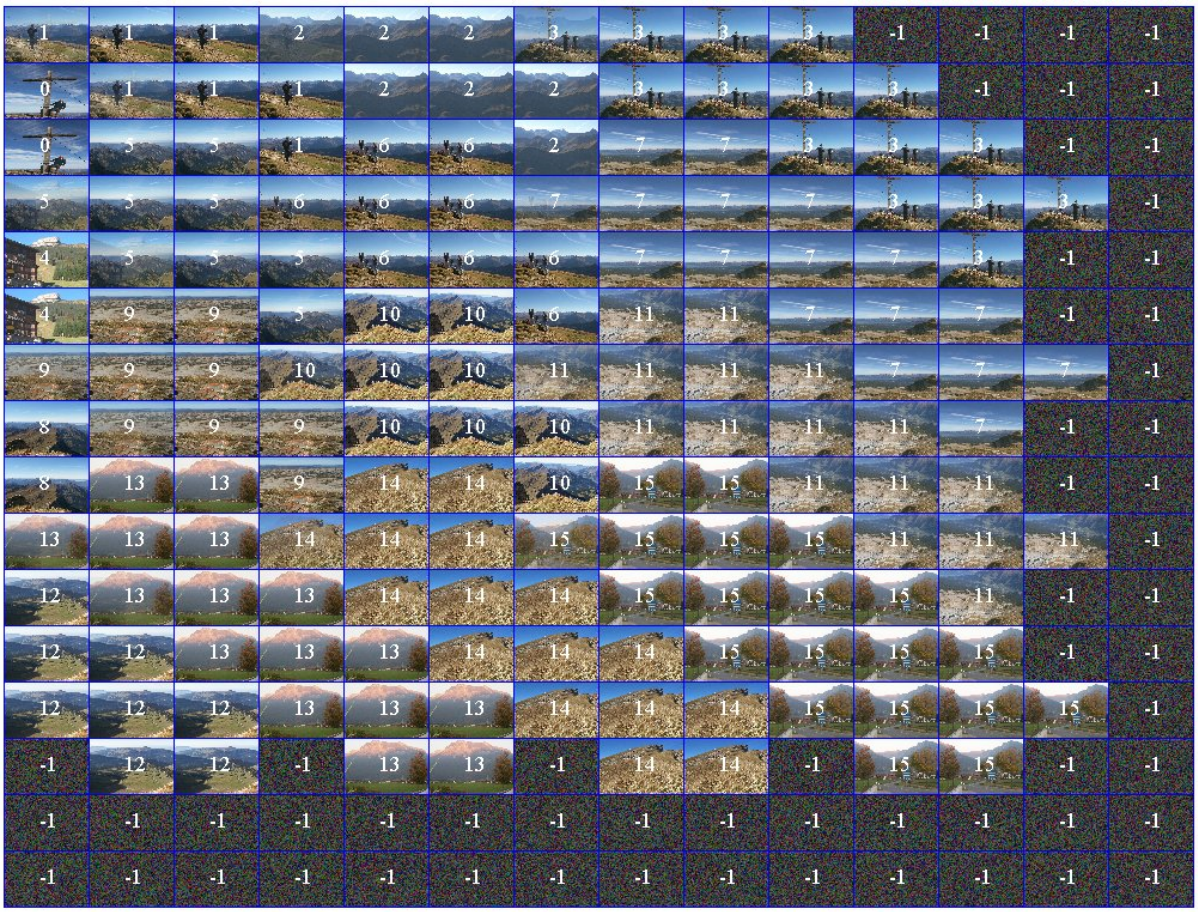


Abb.: 4.13: Stabiler Zustand der Feature Map nach 20 Iterationen

Die Abb.: 4.14 stellt die Ergebnisse für das Aufschalten der Testbilder auf die Feature Map dar. Die linke Spalte enthält die Testbilder und die rechte Spalte das entsprechende gewinnende Element der Feature Map.

Die Testbilder 0,1,2,3 und 5 weisen eine große Ähnlichkeit mit den beiden Gipfelbildern aus dem Trainingsset (siehe Abb. 4.11) auf. Obwohl diese Testbilder im Trainingsset nicht vorhanden sind, erfolgt eine richtige Zuordnung.

Das Bild 4 fehlt ebenfalls im Trainingsset. Das Kohonen-Netz wählt dazu das Trainingsbild der Gesteinsformation aus. Allerdings nimmt die Differenz aus den beiden Pixelbildern einen maximalen Wert an. Daraus lässt nun ableiten, dass es sich beim Testbild 4 um einen Ausreißer handelt. Das ist auch tatsächlich der Fall, denn die Trainingsbilder enthalten keinerlei Gebäudemotive.

Die Testbilder 6 und 9 bis 15 liefern relativ gut übereinstimmende Bilder aus dem Trainingsset. Bei den Bildern 7 und 8 fehlt das entsprechende Bild im Trainingsset. Deshalb gehören diese Bilder zusammen mit dem Bild 4 zu einem eigenen Cluster von Ausreißern.

Abb.: 4.14: Test der Bild-Cluster durch das Aufschalten der Testbilder auf die Feature Map

Kapitel 5

Vorhersage von Zeitreihen

Unter dem Begriff der Vorhersage von Zeitreihen versteht man die Auswertung einer zeitlichen Folge von Daten, die von einem Startzeitpunkt t_1 bis zum momentanten Zeitpunkt t_2 vorliegen. Neben den statistischen Parametern, wie Mittelwert, Varianz und Erwartungswert, gilt das große Interesse vor allem der Vorhersage des zukünftigen Verlaufes des Daten bis zum Zeitpunkt t_3, wobei gilt $t_1 < t_2 < t_3$. Die Daten selbst setzen sich aus einem deterministischen Signal mit einem überlagerten Rauschen zusammen, wobei beide Signaltypen nicht-linear und nichtstationär sein können.

Eine seriöse Vorhersage von Daten erfordert als Grundvoraussetzung einen tatsächlich vorhandenen inneren Zusammenhang der Daten, für den die Autokorrelationsfunktion ein quantitatives Maß im Wertebereich von $\pm 100\%$ liefert. Allerdings ist Vorsicht geboten, denn die von Null verschiedenen Werte der Autokorrelationsfuktion können sich zufällig ergeben und stellen daher keine hinreichende Bedingung für die Existenz eines inneren Zusammenhanges dar.

Entsprechend der Herkunft der Daten unterscheidet man zwischen zwei grundsätzlichen Methoden der Vorhersage von Zeitreihen.

- Die Daten basieren auf bekannten physikalischen Grundgesetzen oder mathematischen Funktionen. In diesem Fall gleicht man die in der Regel stark verrauschten Daten mit geeigneten mathematischen Funktionen mittels einer Regressionsanalyse aus. Z.B eignen sich zur Vorhersage der Position einer bewegten Masse gemäß dem Newtonschen Gesetz Polynome zweiter Ordnung. Der Versuch, bei einem vorhandenen, jedoch mathematisch unbekannten inneren Zusammenhang, trotzdem ein mathematisches Modell anzunehmen, führt bei der Regression zwar zu minimalen Fehlerquadraten, aber die eigentliche Vorhersage der Zeitreihe misslingt.

- Die Daten weisen einen völlig unbekannten inneren Zusammenhang auf, d.h. es existiert kein mathematisches Modell. Unter dieser Voraussetzung wählt man zur Vorhersage von Zeitreihen die Methoden der neuronalen Netze, bei denen die vorliegenden Daten mit einem Trainingsalgorithmus „gelernt“ werden. Bei einer Ein-Schritt-Vorhersage der Daten eignen sich vorwärts gekoppelte neuronale Netzstrukturen. Rückgekoppelte Netzstrukturen ermöglichen auch Mehr-Schritt Vorhersagen. Tyische Beispiele stellen alle Arten von Finanzdaten dar, die sich - falls tatsächlich ein innerer Zusammenhang der Daten existiert - erfolgreich mittels Elman- oder Jordan-Netzen vorhersagen lassen.

Die Bewertung einer Vorhersage basiert auf der normierten Differenz zwischen einem vorhergesagten Signal und dem später tatsächlich eintreffenden Signal.

5.1 Vorhersage mittels nicht-linearer Regression

Eine Vorhersage mittels Regression basiert auf der Extrapolation eines mathematischen Modells, das den inneren Zusammenhang der vorliegenden und zu analysierenden Daten näherungsweise beschreibt. Daher gilt es zuerst, die beiden folgenden Bedingungen zu klären:

- Feststellen, ob ein innerer Zusammenhang überhaupt bestehen kann durch die Berechnung der Auto-Korrelationsfunktion als notwendige Bedingung für die Existenz dieses Zusammenhanges.
- Aufstellen eines mathematischen Modells in Form eines Gleichungssystems, das die physikalischen oder die mathematischen Zusammenhänge, die den Daten zu Grunde liegen, näherungsweise beschreibt.

Beispiel eines simulierten Datensatzes

Das Beispiel veranschaulicht die Vorgehensweise bei der Überprüfung der Voraussetzungen für ein erfolgreiches Regressionsverfahren. Die n Testdaten setzen sich aus einem parabelförmigen Verlauf und einem überlagerten chaotischen Signalverlauf gemäß den Gleichungen 5.1 mit der Taktvariablen k zusammen:

$$\begin{aligned} u_k &= \tfrac{k}{n} + \tfrac{k^2}{n^2}; \\ v_k &= 4 \cdot v_{k-1} \cdot (1.0 - v_{k-1}); \quad v_0 = 0.1 \\ y_k &= u_k + b \cdot v_k; \qquad\qquad b = 0.25 \end{aligned} \tag{5.1}$$

Die Berechnung der Auto-Korrelationsfunktion erfolgt mit den Gl.: 5.2. Dazu teilt man den aus n Daten bestehenden Datensatz in zwei gleiche Teile (g,h) der Länge n/2 und bestimmt für jede der beiden Hälften den Mittelwert (m_g, m_h) und die Varianz (σ_g^2, σ_h^2). Die gesuchte Auto-Korrelationsfunktion ergibt sich dann aus der Auto-Kovarianz, die mittels der Teilvarianzen noch auf den Definitionsbereich von ± 1 zu normieren ist.

$$\begin{aligned} g_k &= y_k \\ h_k &= y_{k+\tau} \\ \\ nn &= \tfrac{n}{2} \\ m_g &= \tfrac{1}{nn} \sum_{k=1}^{nn} g_k \\ m_h &= \tfrac{1}{nn} \sum_{k=1}^{nn} h_k \\ \\ \sigma_g^2 &= \frac{1}{nn} \sum_{k=1}^{nn} (g_k \cdot g_k) - m_g^2 \\ \sigma_h^2 &= \frac{1}{nn} \sum_{k=1}^{nn} (h_k \cdot h_k) - m_h^2 \\ \\ \rho_\tau &= \frac{\frac{1}{nn} \sum_{k=1}^{nn} (g_k \cdot h_k) - m_g m_h}{\sqrt{\sigma_g^2 \sigma_h^2}} \end{aligned} \tag{5.2}$$

Die Abb. 5.1 stellt vor dem Hintergrund der simulierten Daten, die mit blauen Punkten markiert sind, den Verlauf des Betrages der Auto-Korrelationsfunktion für 200 Daten dar. Gesucht ist der weitere Verlauf der Daten im Intervall bis zum Zeittakt von 300. In Übereinstimmung mit der Theorie, beginnt die Auto-Korrelationsfunktion bei 100% und klingt nur bis auf ca. 20% ab, d.h. es liegt hier ein innerer Zusammenhang der entsprechend simulierten Daten vor und die Vorhersage auf der Basis der Regressionsanalyse und des bei der Simulation angesetzen und daher bekannten parabelförmigen zeitlichen Verlaufs der Daten ist sinnvoll.

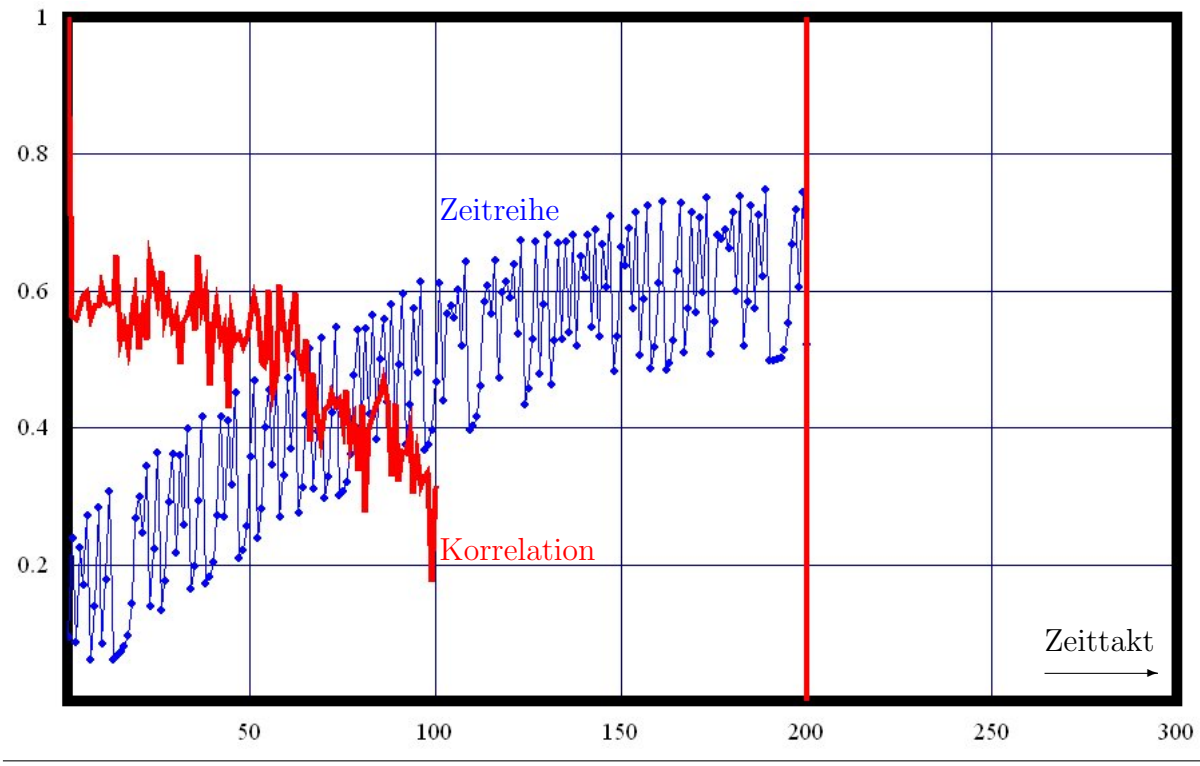


Abb.: 5.1: Verlauf derAuto-Korrelationsfunktion und der simulierten Daten

Die nächste Aufgabe besteht darin, ein geeignetes mathematisches Modell aufzustellen, das den inneren Zusammenhang der Zeitfolge beschreibt. Bei den simulierten Daten, auf denen die obige Abb. : 5.1 basiert, eignet sich als Modell die allgemeine Gleichung einer Parabel, deren Koeffizienten die Regressionsalgorithmen liefern.

Für den häufigen Fall, dass die Vorhersagewerte einen zu hohen Fehler aufweisen, kann man mittels der Autokorrelationsfunktion die Länge des Intervalls des Regressionsverfahrens anpassen oder auch und mehrere konkurrierende mathematische Modelle einsetzen. Bei den einzelnen Datensätzen empfiehlt sich eine Gewichtung, die sich am Erwartungswert der Daten orientiert.

Eine weitere hilfreiche Maßnahme besteht in der Erweiterung der Regressionsalgorithmen um einen Stabilitätsfaktor ϵ entsprechend der Gl. 5.3 zur Steigerung der Robustheit der Matrizeninversion.

$$(H^T H)^{-1} \quad \longrightarrow \quad (H^T H + \epsilon I)^{-1} \tag{5.3}$$

Die Gl. 5.4 beschreibt die Qualität einer Vorhersage durch den Integritywert. Die Integrity nimmt den Wert 100% für den Fall an, dass die Vorhersage mit dem tatsächlichen Wert übereinstimmt. Als normierender Maximalwert dient der Betrag des Maximalwertes der vorliegenden Daten.

$$\text{Integrity} = \left(1 - \frac{|\text{Differenz}|}{|\text{Maximalwert}|}\right) \cdot 100\%; \quad -100\% \leq \text{Integrity} \leq 100\% \tag{5.4}$$

5.2 Beispiel einer Regression

Das mathematische Modell für die in diesem Beispiel simulierten Daten lautet:

$$y_{Modell} = a + b \cdot k + c \cdot k^2 \tag{5.5}$$

Für jeden Wert y der Daten sollte diese Gl.: 5.5 näherungsweise gelten. Somit lässt sich ein überbestimmtes lineares Gleichungssystem für die vorliegenden 200 Daten aufstellen, das in Form einer Matrizen-Vektor Darstellung wie folgt lautet und nach a,b,c aufzulösen ist:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & \vdots & \vdots \\ 1 & 200 & 200^2 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{200} \end{pmatrix} \tag{5.6}$$

Die Lösung eines überbestimmten Gleichungssystems gelingt mit der klassischen Methode der kleinsten Fehlerquadrate, bei der die quadratische Betragssumme der einzelnen Abweichungen zwischen den Funktionswerten des mathematischen Modells und den tatsächlich vorliegenden Daten gebildet und anschließend mininmiert wird. In der Matrizen-Vektor-Schreibweise läuft dies auf die Inversion einer quadratischen Matrix hinaus, wobei die Ordnung der Matrix der Anzahl der unbekannten Koeffizenten entspricht.

Ordnet man dem mathematischen Modell die rechteckige Matrix H zu, bringt die einzelnen zu bestimmenden Koeffizienten in einem Vektor a unter und fasst die Daten in einem Vektor y zusammen, dann gilt allgemein:

$$H \cdot a = y \tag{5.7}$$

Die Gl. 5.7 lässt sich von links mit der transponierten H-Matrix multiplizieren. Dadurch entsteht eine quadratische Matrix $H^T H$, die invertierbar ist. Die Matrix $P^{\#}$ bezeichnet man anschaulich als pseudo-inverse Matrix.

$$H^T H \cdot a = H^T \cdot y \tag{5.8}$$
$$a = (H^T H)^{-1} \cdot H^T \cdot y \tag{5.9}$$
$$P^{\#} = (H^T H)^{-1} \cdot H^T \tag{5.10}$$
$$a = P^{\#} \cdot y \tag{5.11}$$
$$a = P^{\#} \cdot y \tag{5.12}$$

Die Vorhersage der Daten erfordert nur noch die Extrapolation des mathematischen Modells, dessen Koeffizienten a sich nach der Multiplikation der Pseudo-Inversen mit dem Vektor y der vorliegenden Daten entsprechend der Gl. 5.12 berechnen lassen.

Für die vorliegenden simulierten Daten ergibt sich nach der Auswertung der Gl.: 5.13 eine Vorhersage z,

$$z = a_1 + a_2 \cdot k + a_3 \cdot k^2; \quad k > 200; \tag{5.13}$$

die zusammen mit den gegebenen Daten und dem Verlauf des mathematischen Modells in der Abb.: 5.2 graphisch dargestellt ist. Die Vorhersage (rot) erkennt, dass die Werte der zukünftigen Daten abnehmen, was durch das Betrachten der gegebenen Daten (blau) nicht

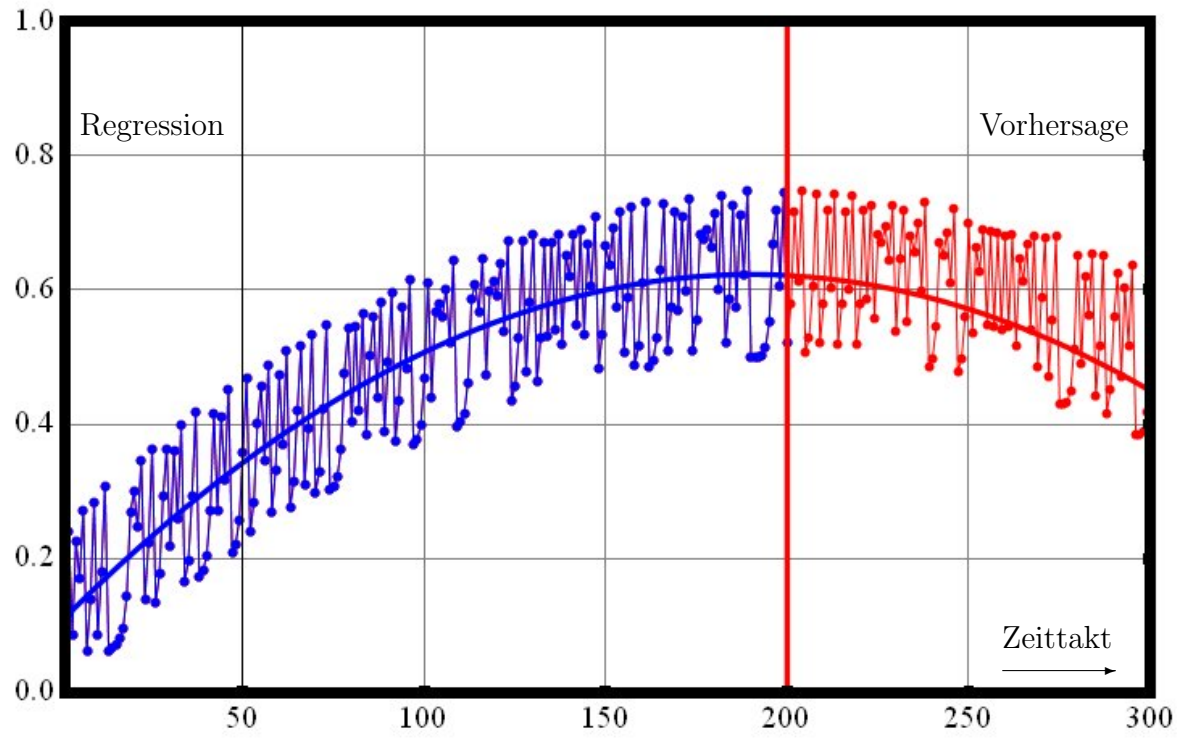


Abb.: 5.2: Vorhersage von simulierten Daten auf der Basis von 200 Datensätzen

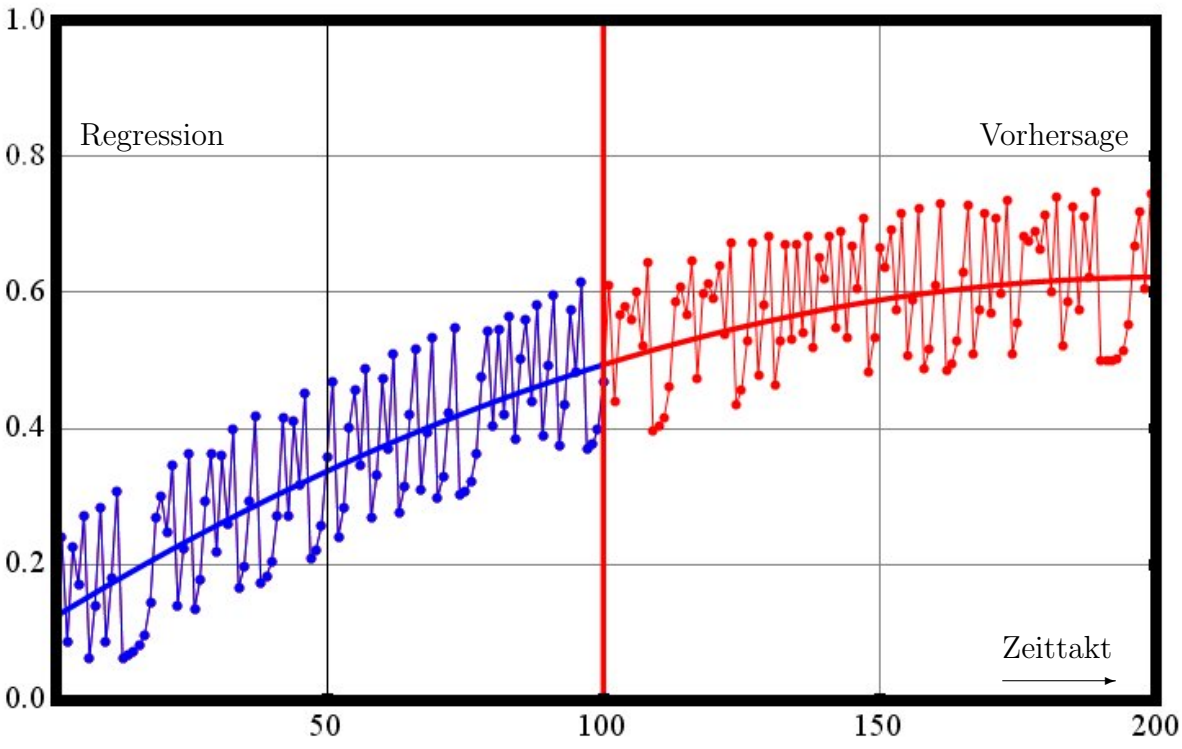


Abb.: 5.3: Vorhersage von simulierten Daten auf der Basis von 100 Datensätzen

erkennbar ist. Falls es sich bei den simulierten Daten um Kursschwankungen gehandelt hätte, wäre diese Vorhersage sehr hilfreich gewesen.

In der Abb.: 5.3 wurde das Regressionsintervall halbiert. Trotz der damit verbundenen Halbierung der Datenbasis ermöglicht das Verfahren eine exakte Vorhersage des Zeitverlaufes.

Schaltet man die Simulation der Daten auf einen periodischen zeitlichen Verlauf um und behält trotzdem das mathematische Modell einer Parabel bei, dann treten, wie in der Abb.: 5.4 dargestellt, in beiden Fällen trotz der nach wie vor vorhandenen Autokorrelation falsche Vorhersagewerte auf, wobei die größere Datenbasis von 200 Daten besonders fehlerhafte Vorhersagen produziert.

Die Abb.: 5.4 veranschaulicht deutlich die ganze Problematik, die bei der Vorhersage auf der Basis eines Regressionsverfahrens auftritt. Wenn man das mathematische Modell des Verlaufs von Zeitreihen nicht wenigstens näherungsweise kennt oder davon ausgehen muß, dass sich das Modell in Abhängigkeit von der Zeit verändert, dann sollte man ein Regressionsverfahren zur Vorhersage von Zeitreihen meiden.

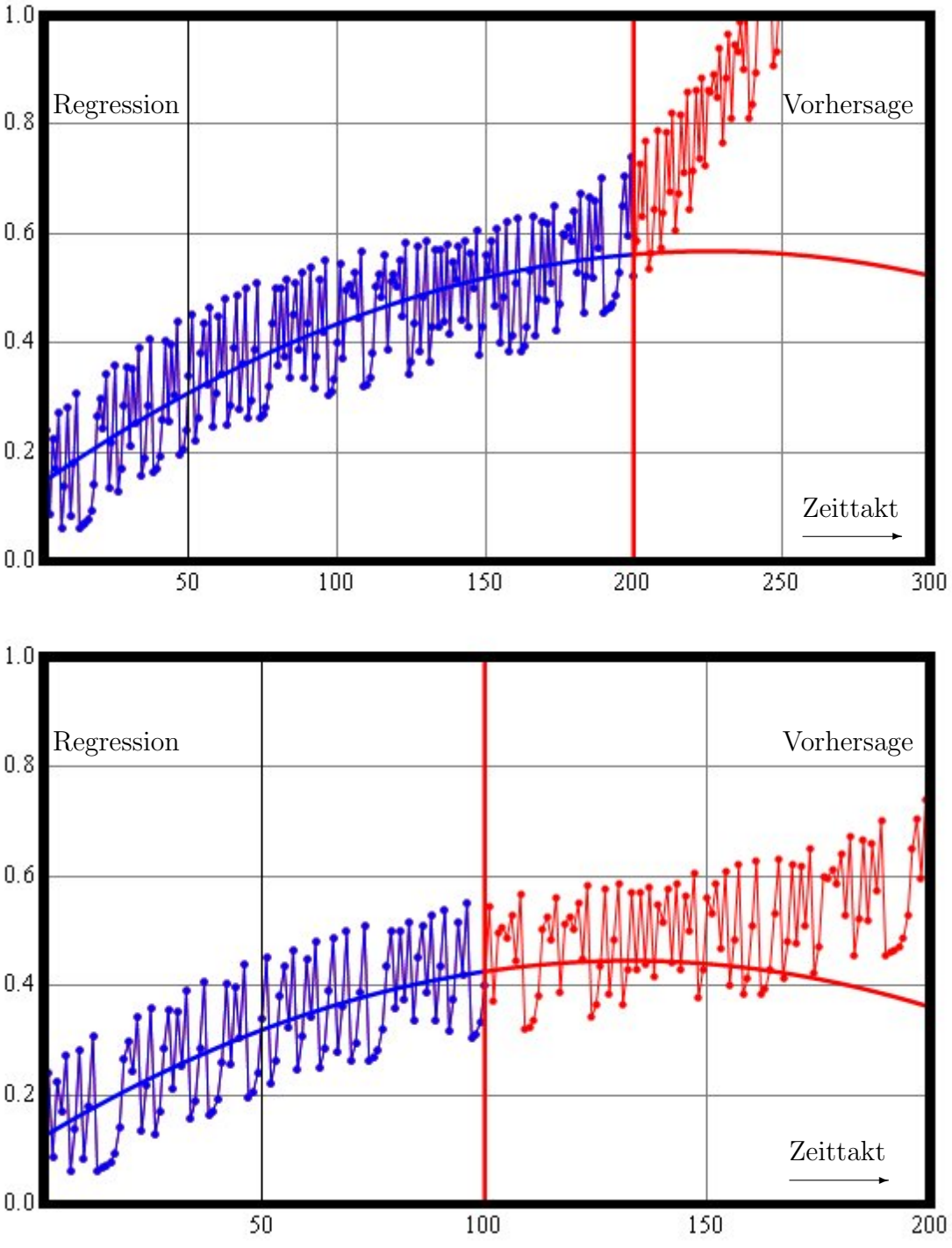


Abb.: 5.4: Falsches mathematisches Modell für 200 (oben) bzw. 100 (unten) Regressionsdaten

5.3 Vorhersage mit neuronalen Netzen

Neuronale Netze eignen sich besonders gut zur Vorhersage von Zeitreihen. Im Gegensatz zu den Regressionsverfahren, die stets die Annahme eines mathematischen Modells erfordern, „lernt“ das neuronale Netz die inneren Zusammenhänge in den Datensätzen durch die Konvergenz eines geeigneten algorithmischen Verfahrens.

Als neuronale Netzstruktur zur Datenvorhersage empfiehlt sich grundsätzlich ein rückgekoppeltes Netz, mit dem sich dynamische Ausgangssignalfolgen erzeugen lassen. Im Hinblick auf die numerische Berechenbarbeit der Gewichte des neuronalen Netzes und die erforderliche rasche Konvergenz der Trainingsalgorithmen eignen sich vor allem jene neuronalen Netze, die in der Literatur als Jordan- oder Elman-Netze bekannt sind und die in vielfältigen praktischen Anwendungen ihre Tauglichkeit erfolgreich unter Beweis stellen.

Jordan- oder Elman-Netze basieren auf vorwärts gekoppelten Perceptron-Netzen, die aus einer Eingangs- Zwischen- und Ausgangsschicht bestehen und bei denen die Ausgangssignale der Ausgangsschicht (Jordan-Typ) bzw. der Zwischenschicht (Elman-Typ) zusätzlich zu den Neuronen in der Eingangsschicht gelangen. Durch das Setzen aller Rückkopplungsgewichte auf Eins, entfällt das Trainieren dieser Gewichte und der bekannte Backpropagation-Algorithmus, der eigentlich nur für reine Perceptron-Netze gilt, lässt sich auf diesen Sonderfall eines rückgekoppelten Netzes mit geringen Anpassungen anwenden.

Die Abb.: 5.5 stellt die Struktur eines speziellen Netztyps dar, der sich aus dem Jordan- und dem Elman-Netz zusammensetzt. In diesem Fall gelangen die Ausgangssignale der Zwischenschicht, ebenso wie die Signale der Zwischenschicht, auf den Eingang der Eingangschicht. Das Netz selbst besteht aus zwei Eingangs-, drei Zwischen- und zwei Ausgangsneuronen. Der jeweils oberste Eingang der beiden Eingangsneuronen nimmt den Zeittakt auf. Dann folgen die eigentlichen Eingangssignale, die den Abtastwerten des Trainingsdatensatzes entsprechen. Die Struktur dieses Netzes ist durch die folgenden Parameter festgelegt:

$$
\begin{array}{lll}
\mathrm{L} & \rightarrow & \text{Anzahl der Ausgangsneuronen,} \\
 & & \text{Länge des Vorhersage-Intervalles und} \\
 & & \text{Anzahl der Datenkanäle} \\
\mathrm{N} & \rightarrow & \text{Anzahl der Eingangsneuronen} \\
\mathrm{K} & \rightarrow & \text{Anzahl der Neuronen der Zwischenschicht} \\
\text{Gewichte} & \rightarrow & 2(1+L+L+K)+K\cdot L+L\cdot K
\end{array}
\tag{5.14}
$$

Liegen 10 Trainingsdaten vor, dann lautet die Folge der Indizes z.B. für zwei Eingangssignale:

$$
\begin{array}{lllllll}
\mathit{Inputkanal} & \rightarrow & (1,2), & (3,4), & (5,6), & (7,8) \\
\mathit{Outputkanal} & \rightarrow & (3,4), & (5,6), & (7,8), & (9,10)
\end{array}
\tag{5.15}
$$

Diese Art der Signalaufschaltung bedeutet, dass ein ganzes Intervall von Signalen an den Eingang des Netzes gelangt und dann nach Abschluss der Trainingsphase am Ausgang des Netzes das Intervall der zukünftigen Signale erscheinen soll. Die restlichen Signale der Eingangsneuronen stammen aus der rückgekoppelten Zwischen- bzw. Ausgangschicht.

Die Recallphase dient zur Vorhersage der Zeitreihe und erfordert mehrere Durchläufe, damit sich ein stabiles Ausgangsignal einstellt. Am Ausgang des neuronalen Netzes liegt dann das vorhergesagte Intervall aus L Signalen an.

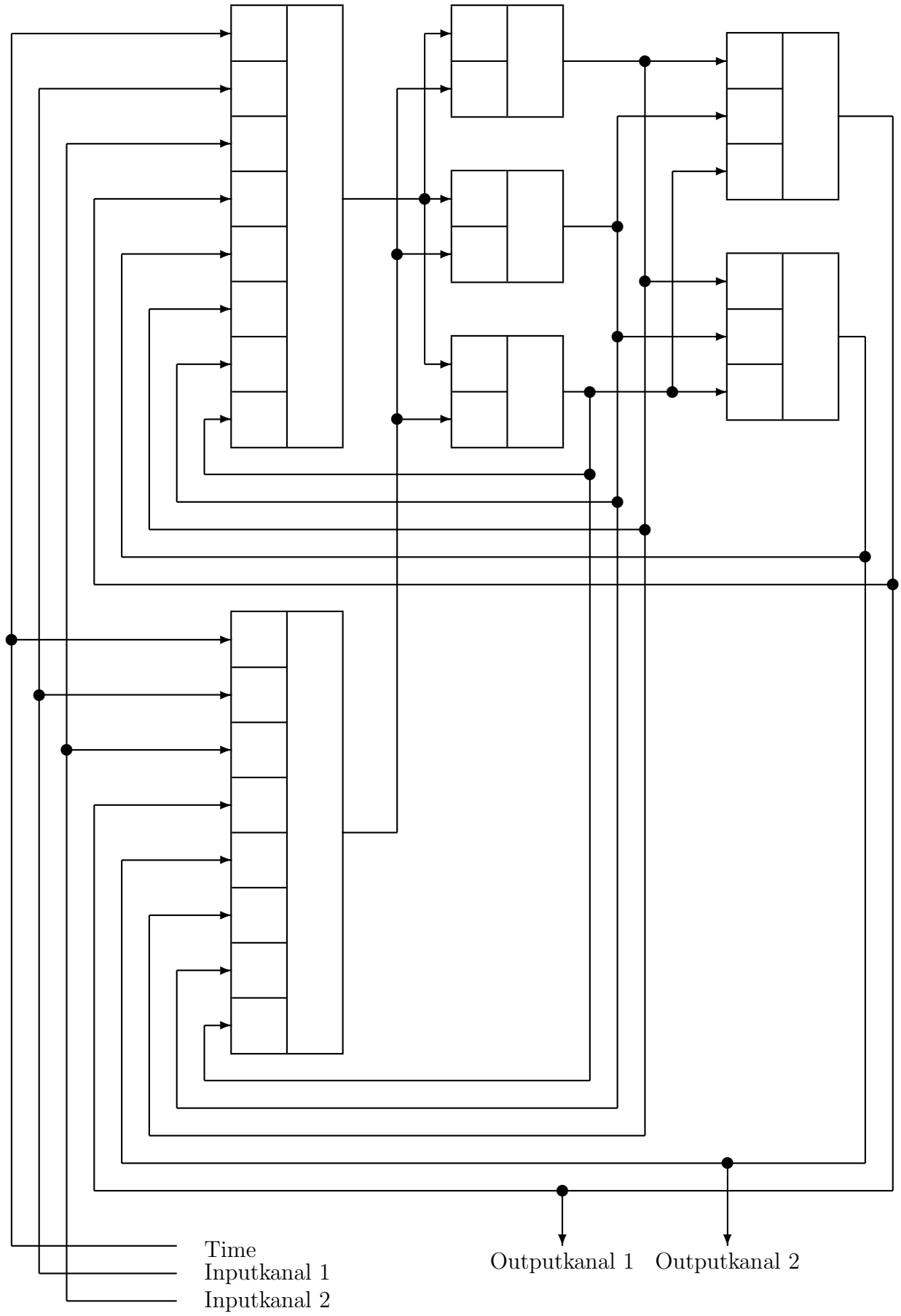


Abb.: 5.5: Struktur des rückgekoppelten neuronalen Jordan-Elman Netzes

Das Trainieren der Gewichte basiert auf der Methode des überwachten Lernens („supervised training“). Dazu setzt man zu Beginn per Zufallsgenerator die Gewichte, legt die Eingangssignale an das Netz, berechnet die Ausgangssignale und bildet anschließend die Differenz zu den gewünschten Ausgangssignalen. Diese Differenz entspricht dem Fehler des Netzes. Schließlich verändert man die Gewichte so, dass der Fehler gegen Null konvergiert. Dann beginnt ein neuer Rechenlauf, den man als Trainieren bezeichnet.

Das Verändern der Gewichte, also das Lernen, ist allerdings nicht so einfach, denn es sind bei einer Anzahl von m Eingangssignalen eine Anzahl von m Ausgangsmustern zu trainieren. Hat z.B. das Netz das erste Muster vollständig gelernt, dann darf es beim Trainieren des zweiten Musters auf keinen Fall das erste Muster wieder wegtrainieren (vergessen). Deshalb trainiert man die gewünschten Ausgangssignale nur teilweise und sequentiell an und wiederholgt die Trainingsläufe zyklisch.

Das Verändern der Gewichte basiert wieder auf dem biologischen Modell des Lernens. Gemäß der Lernregel nach Hebb gilt:

$$\Delta w_{i,j} = \gamma \cdot \text{output}_i \cdot \text{output}_j \qquad 0 \leq \gamma \leq 1 \tag{5.16}$$

Die Änderung ist mit einem Faktor γ proportional dem Produkt aus dem Ausgangssignal und dem Eingangssignal des Neurons. γ stellt die Lernrate dar und sollte zwischen 0 und 1 liegen.

Dieser Grundalgorithmus gemäß der Gleichung (5.16) dient nun als Ausgangspunkt für das Backpropagation-Verfahren. Dazu führt man einen Fehlerterm δ für jedes einzelne Neuron ein. Ausgehend vom Fehler der Ausgangsschicht schreitet man rückwärts durch das neuronale Netz und schreibt (propagiert) den Fehler bis auf die Eingangsschicht zurück. Daher stammt die Bezeichnung „Backpropagation“. Schließlich multipliziert man die Änderung der Gewichte noch mit der Ableitung der Sigmoid-Funktion, um die Konvergenz des Verfahrens zu beschleunigen.

Die Konvergenz dieses Verfahrens ist in jedem Anwendungsfall zu überprüfen. Falls das neuronale Netz schlecht dimensioniert ist oder falls die zu trainierenden Muster sich zu sehr ähneln, gefährdet dies durch das Auftauchen von Nebenminima die Konvergenz. Die Lösung besteht darin, durch eine entsprechende Anzahl von Testläufen die optimale Neuronenzahl zu bestimmen und insgesamt eine optimale, d.h. auf das zu lösende Problem angepasste, Netzkonfiguration zu entwickeln.

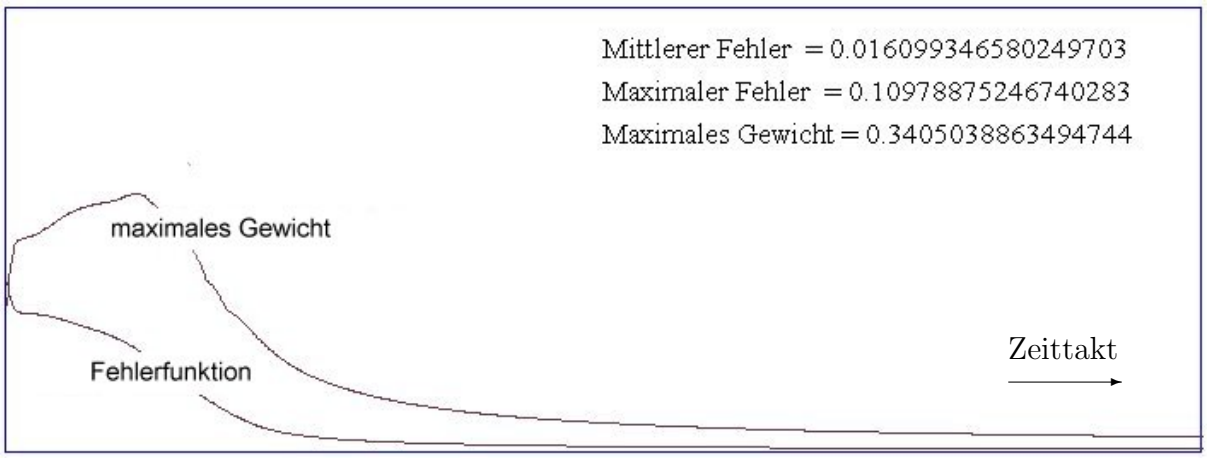


Abb.: 5.6: Konvergenz der Backpropagation-Fehlerkurve als Funktion der Trainingszyklen

Während der Trainingsphase berechnet man die betragmäßige Fehlersumme aus den Differenzen der tatsächlichen Ausgangssignale und der gewünschten Ausgangssignale. Das

Beobachten dieser Fehlerfunktion sowie weiterer Parameter liefert, wie in der Abb.: 5.6 zu erkennnen, wertvolle Hinweise über den Zustand des neuronalen Netzes in Abhängigkeit von den Trainingszyklen.

Der Algorithmus sei am Beispiel des zufällig ausgewählten Gewichtsfaktors $w_{5,7}$ konkret erläutert. Dazu wertet man die Gleichungen (5.17) bis (5.21) in der angegebenen Reihenfolge aus und beginnt dann wieder mit der ersten Gleichung (5.17).

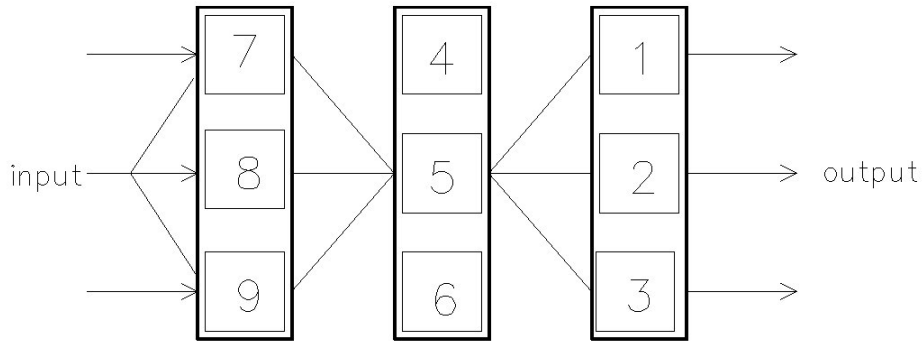


Abb.: 5.7: Beispielhafter Verlauf des Backpropagation-Verfahrens

Das folgende Gleichungssystem listet die Gleichungen für den Backpropagation-Algorithmus auf.

$$NettoInput_2 = output_4 \cdot weight_{2,4} + output_5 \cdot weight_{2,5} + output_6 \cdot weight_{2,6} \quad (5.17)$$

$$\delta_2 = (output_2 - referenz_2) \cdot sigmoid\,'(NettoInput_2) \quad (5.18)$$

$$\delta_5 = sigmoid\,'(NettoInput_5) \cdot (\delta_1 \cdot weight_{1,5} + \delta_2 \cdot weight_{2,5} + \delta_3 \cdot weight_{3,5}) \quad (5.19)$$

$$\Delta weight_{5,7} = \gamma \cdot \delta_5 \cdot output_7 \quad (5.20)$$

$$weight_{5,7}^{neu} = weight_{5,7}^{alt} + \Delta weight_{5,7} \quad (5.21)$$

Als Abbruchkriterium dient eine Fehlerschranke, welche die minimale Übereinstimmung zwischen den output Signalen des Netzes und den zu trainierenden Mustern definiert. Zur Einsparung von Rechenzeit kann man den analytischen Zusammenhang zwischen den Sigmoid-Funktionen und ihren Ableitungen nutzen

$$\frac{1}{1+e^{-x}} \longrightarrow sigmoid\,'(netto) = sigmoid(netto) \cdot (1 - sigmoid(netto)) \quad (5.22)$$

$$\tanh(x) \longrightarrow \tanh\,'(netto) = 1 - \tanh^2(netto) \quad (5.23)$$

und die Sigmoid-Funktion und ihre Ableitung in Potenzreihen mit wenigen Koeffizienten entwickeln.

Nach Abschluss der Trainingsläufe ist das Netz in der Lage, für ein Intervall von Eingangsdaten das zukünftige Intervall der Daten in Form der als Ausgangssignal vorher zu sagen.

Zur adaptiven Anpassung des Elman- oder Jordan-Netze speichert man während der Recallphase die aktuellen Datensätze und startet bei einer nachlassenden Integrity parallel einen neuen Backpropagation-Trainingslauf. Mit dieser Technik reagieren diese neuronalen Netze auf Änderungen der inneren Struktur der Zeitreihen.

5.4 Beispiele neuronaler Vorhersagen

Vorhersage einer periodischen Funktion

Bei der Vorhersage einer periodischen Funktion handelt es sich um einen einfachen Fall. Das neuronale Netz arbeitet mit 200 Trainingsdaten und besteht aus 5 Eingangs-, 10 Ausgangsneuronen und sagt den Verlauf der periodischen Funktion für 50 Zeittakte voraus. Bereits nach 11 Schritten konvergiert der Backpropagation-Algorithmus und der vorhergesagte Verlauf stimmt mit dem tatsächlichen Verlauf überein. Die besondere Leistung dieses Netzes besteht darin, dass keinerlei Annahmen über das mathematische Modell erforderlich waren.

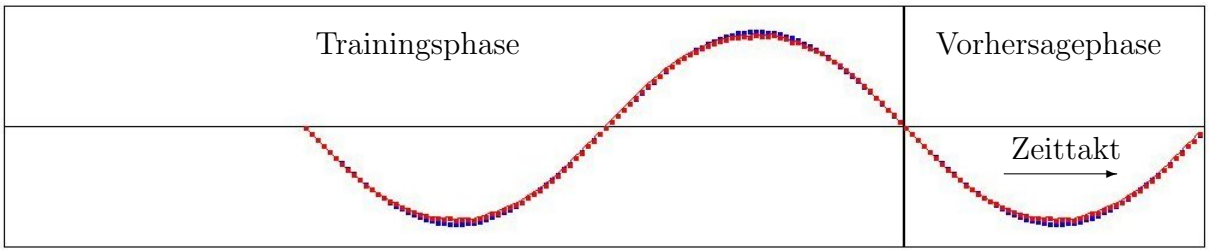


Abb.: 5.8: Vorhersage einer periodischen Funktion

Die rote Punktfolge in der Abb.:5.8 markiert die trainierte bzw. vorhergesagte Funktion und die blauen Punkte entsprechen dem Trainingsdatensatz.

Setzt sich das simulierte Trainingssignal aus einer Grundschwingung und zwei Oberwellen zusammen, dann lässt sich mit diesem neuronalen Netz auch nach 100 Epochen kein zufrieden stellendes Ergebnis erzielen (Abb.:5.9).

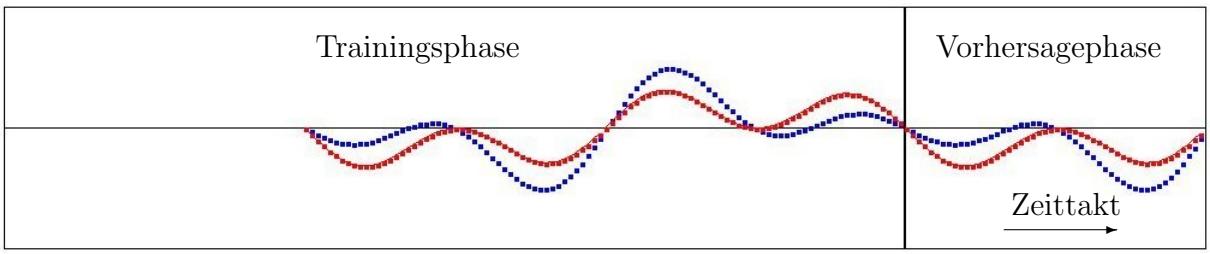


Abb.: 5.9: Vorhersage einer Grundschwingung mit zwei Oberwellen

Die höhere Dynamik des Trainingssignales erfordert nämlich auch die Erhöhung der Anzahl der Neuronen. Mit N = 30 und K=60 gelingt wieder eine Vorhersage, wie in der Abb.: 5.10 zu erkennen ist.

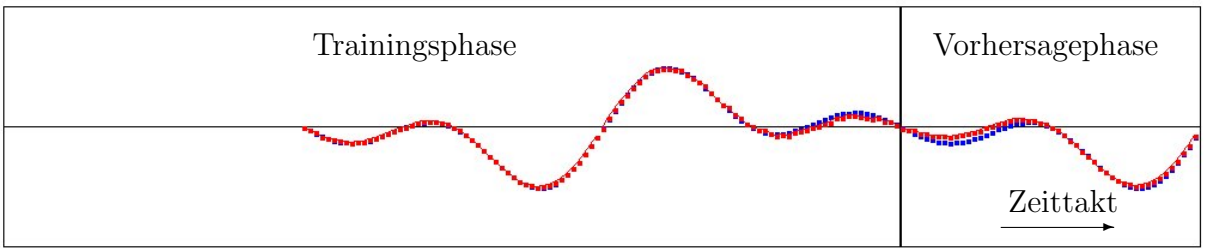


Abb.: 5.10: Vorhersage mit einer Grundschwingung mit zwei Oberwellen mit erhöhter Neuronenanzahl

Vorhersage einer Pseudo-Zufallszahlenfolge

Eine besondere Herausforderung für jedes Vorhersagesystem stellt die Extrapolation von Pseudo-Zufallszahlen dar. Die Abb.: 5.11 zeigt die Ergebnisse des neuronalen Netzes, das in diesem Fall aus N=L=50 Eingangsneuronen und ebenfalls 50 Neuronen in der Zwischenschicht besteht. Gegeben sind 100 Zufallzahlen, von denen die ersten 90 als Trainingsmenge dienen und die restlichen 10 Zahlen die Referenz für die Vorhersage bilden. Der erzielte Vorhersagefehler von 33% liegt deutlich besser als der rein statistische 50% Wert, der sich im Falle einer zufälligen Vorhersage ergeben würde.

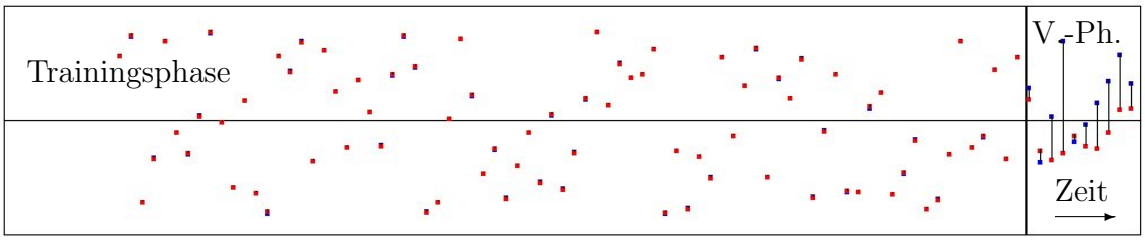


Abb.: 5.11: Vorhersage eines Zufallsgenerators

Die Tab.:5.1 stellt die vorhergesagten und die tatsächlichen Zufallszahlen gegenüber.

Nummer	Vorhersage	Zufallszahl	Fehlerbetrag
91	20	30	10
92/1	-25	-35	10
93/2	-33	5	38
94/3	-27	71	98
95/4	-12	-17	5
96/5	-21	-2	19
97/6	-23	17	40
98/7	-9	36	45
99/8	11	59	48
100/9	12	34	22

Tab.: 5.1: Vorhersage von Zufallszahlen

Die Abb.: 5.12 lässt den Verlauf des über alle 80 Trainingsdaten gemittelten Fehlers am Ende jeder Epoche der Trainingsphase erkennen. Nach 600 Trainingszyklen geht der gerundete Fehlerwert auf Null, d.h. die vorhergesagten Zahlen (rote Punkte) und die tatsächlichen Zufallszahlen der Trainingsmenge (blaue Punkte) stimmen überein. Der dargestellte Fehlerverlauf zeigt, dass trotz der auftretenden relativen Nebenminima der Backpropagation-Algorithmus wegen der programmierten automatischen Steuerung der Lernrate gemäß der Gl.: 5.24 bis zu einem Fehler von 0% konvergiert.

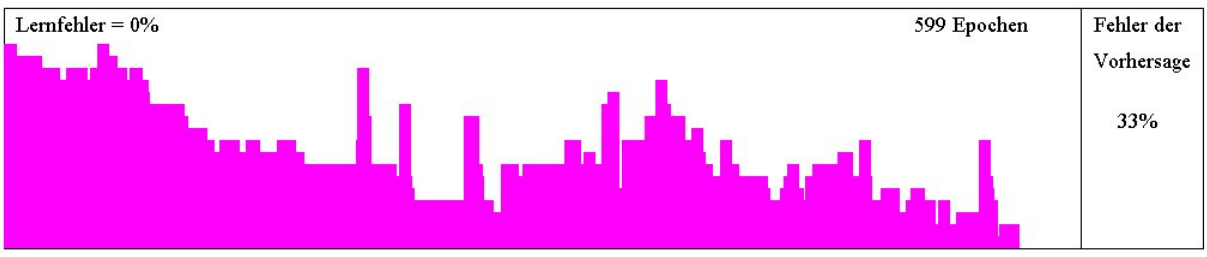


Abb.: 5.12: Konvergenz der Vorhersage eines Zufallsgenerators

$$\text{Lernrate} = \text{Anfangswert der Lernrate} \cdot \left(1 - e^{-\frac{\text{Betrag des Fehlers}}{\text{Zeitkonstante}}}\right) \tag{5.24}$$

Kapitel 6

Zusammenfassung und Ausblick

Der vorliegende Forschungsbericht dokumentiert Ergebnisse, die in der Abteilung „Informatik“ der Fakultät „Wirtschaft und Informatik“ der Fachhochschule Hannover im Sommersemesters 2007 im Rahmen eines von der Forschungskommission genehmigten Forschungssemesters vom Autor erarbeitet wurden. Der Bericht stellt die wichtigsten Verfahren aus dem Gebiet des Data Minings vor, diskutiert auf der Basis von selbst programmierten Beispielen die Eignung der verschiedenen Verfahren für die entsprechenden Anwendungsgebiete und vergleicht die speziellen Eigenschaften der entsprechenden Algorithmen.

Zusammenfassend stellt die Tab.: 6.1 die wichtigsten Verfahren und ihre besonderen Eigenschaften gegenüber:

	Anwendungsgebiete	Verfahren
Warenkorb-Analyse	Versandhandel, Einzelhandel, Auktionen im Web	Logische Warenkörbe Apriori-Verfahren Support und Konfidenz
Klassifizieren	Kundenmanagment, Kreditvergabe, Medizintechnik	Entscheidungsbaum Regression, Support Vektoren Neuronale Netze
Clusterbildung	Aufspüren von abweichendem Kundenverhalten	Abstandsverfahren Neuronales Kohonen-Netz mit Feature Map
Anwendungsgebiet	Vorhersage von Preisen, Umsätzen, Aktienkursen, usw.	Regression Neuronale Elman- und Jordan-Netze

Tab.: 6.1: Zusammenfassung der Mining Mining Verfahren

Der Anwender eines Data Mining Verfahrens sieht sich in der Regel mit der Problematik konfrontiert, auf einen vorgegebenen Datenbestand ein Data Mining Verfahren anzuwenden und dazu dann einen geeigneten Algorithmus aus einer Reihe von Möglichkeiten auszuwählen. Die im vorliegenden Forschungsbericht dokumentierten Erkenntnisse

unterstützen den Anwender bei dieser Entscheidung, die vor allem von der gegebenen Aufgabenstellung und den wesentlichen Parametern des gegebenen Datenbestandes abhängt.

Die folgenden Parameter wirken sich besonders stark auf die Auswahl eines algorithmischen Verfahrens aus:

- Anzahl der verfügbaren Datensätze
- Anzahl der relevanten Attribute eines Datensatzes
- Anzahl der möglichen Klassen eines Datensatzes
- Herkunft der Daten (Fragebogen, Messwerte, Medizindaten)
- Eindeutige oder widersprüchliche Daten
  (unterschiedliches Kundenverhalten bei identischen Attributen)

Der Forschungsbericht geht im Zusammenhang mit den dokumentierten Verfahren auf diese Parameter ein und empfiehlt geeignete Algorithmen.

Die Fortführung der Forschungsarbeiten wird sich mit der optimalen Kombination der in diesem Bericht dokumentierten Basisverfahren beschäftigen. So lässt sich z.B. das ohnehin schon enorm leistungsfähige Support Vektor Verfahren durch ein nachgeschaltetes neuronales Netz auch auf Datenbestände mit vielen Klassen erfolgreich anwenden.

Im Bereich der Clusterbildung weisen 3D-Feature Maps auf die zukünftige Entwicklung bei der automatisierten Suche in großen Datenbeständen nach „Gold Nuggets“ oder nach „Black Nuggets“ hin. Im ersten Fall erschließen sich neue Kundenstämme oder Märkte und im zweiten Fall lassen sich kriminelle Aktivitäten aufspüren.

Eine erhöhte Integrity darf man von der Erweiterung der neuronalen rückgekoppelten Netze um adaptive Komponeten erwarten. Die laufende Anpassung der Algorithmen in Abhängigkeit von der Differenz aus prognostiziertem Verlauf und tatsächlich eingetretenem Verlauf realisiert ein adaptives Verhalten. Damit gelingt es zukünftige Entwicklungen besser als bisher zu prognostizieren. In den Bereichen mit großem Finanzvolumen genügt schon eine geringe prozentuale Verbesserung des Integritywertes, um beständig Gewinne zu erwirtschaften.

Künftig wird die Datenflut noch weiter ansteigen, z.B. durch das interaktive Ausfüllen von Web-Formularen, das automatische Einsammeln und Parsen von Web-Seiten und das computergestützte Erfassen aller medizinischen Daten. Dabei handelt sich beim Parsen um eine schwierige Aufgabe, denn die Daten liegen als Web-Seite in unstrukturierter Form vor, werden jedoch in strukturierter Form, z.B. als Tabelle, zur Weiterverarbeitung benötigt.

Data Mining Verfahren dienen primär der Beschaffung von Wettbewerbsvorteilen und dienen somit zur Umsatz- und Gewinnsteigerung eines Unternehmens. Daher besteht von Seiten der Unternehmen ein großes Interesse am Einsatz dieser Verfahren.

Allerdings lassen sich diese Verfahren auch zum Ausspionieren von Kundendaten benutzen und in diesem Zusammenhang beschreibt der Ausdruck „gläserner Kunde“ die Gefahren, welche moderne und leistungsfähige Data Mining Verfahren darstellen können.

Jeder Kunde sollte sich genau überlegen, welche Informationen er über seine Person und sein Kaufverhalten „preisgibt“, um damit der missbräuchlichen Verwendung seiner Daten, wie z.B. durch den nicht genehmigten Weiterverkauf seiner persönlichen Angaben, vorzubeugen.

Kapitel 7

Anhang

Abbildungsverzeichnis

Literatur: Grundlagen Data Mining

[1] Berry M., Linoff G. (2003): *Data Mining Techniques*, John Wiley & Sons, ISBN 0471482994, 2003

[2] Han Jiawie, Kamber Micheline (2006): *Data Mining, Concepts and Techniques*, Second Edition, Morgan Kaufmann ISBN-13:978-1-55860-901-3

[3] Witten I., Frank E. (2005): *Practical Machine Learning Tools and Techniques with Java implementations*, Morgan Kaufmann, ISBN 0-12-088407-0

[4] Peterson H. (2005): *Data Mining, Verfahren, Prozesse, Anwendungen*, Oldenburg Verlag, ISBN-978-3-486-57715-0

[5] Neusser K., (2006): *Zeitreihenanalyse in den Wirtschaftswissenschaften*, Teubner-Verlag, ISBN-13 978-3-8351-0117-3

[6] Hamm C. (2007): *Oracle Data Mining*, Oracle In-focus Series 25, ISBN 0-9744486-3-X

[7] Pang-Ning T., Steinbach M:, Kumar V. (2006): *Introduction to Data Mining*, John Wiley & Sons, ISBN 0-321-32136-7, 2006

[8] Nasraoui O. (2006): *Advances in Web Mining and Web Usage Analysis*, 8th international workshop on the Web, WebKDD 2006 Philadelpha, Springer Verlag, ISBN 3-540-77484-X

[9] Apolloni B. (2007): *Knowledge-based intelligent information and engineering systems*, Springer Veralg, ISBN 3-540-74828-8

[10] Orgun M. (2007): *Advances in artificial intelligence*, 20th Conference on Artificial Intelligence, Springer Verlag, ISBN 3-540-76926-9

[11] Buck W. (2007): *SQL Server 2005: das Handbuch für Administratoren*, Addison Wesley, ISBN 3-8273-2663-X

[12] Klünter D. (2008): *LDAP verstehen, OpenLDAP einsetzen: Grundlagen und Praxiseinsatz*, dpunkt-Verlag, ISBN 3-89864-263-1

[13] Hertzberg J. (2007): *Advances in artificial Intelligence: proceedings*, German Conference on AI in Osnabrück, Springer Verlag, ISBN 3-540-74564-5

[14] Alhajj R. (2007): *Advanced data mining and applications*, 3th international conference, ADMA 2007, Harbin, China, Springer Verlag, ISBN 3-540-73870-3

[15] Kok, J. (2007): *Knowledge discovery in databases*, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw 2007, Springer Verlag, ISBN 3-540-74975-6

Literatur: Algorithmen

[16] Agraval R., Srikant R. (1994): *Fast Algorithms for Minining Association Rules*, San Jose, USA 1994, http://www.almaden.ibm.com/cs/people/ragrawal/papers/vldb94-rj.ps(29.10.2001)

[17] Vapnik V. (1995): *The Nature of Statistical Learning Theory*, Springer, N.Y., ISBN 0-387-94559-8.

[18] Gunn S. (1998): *Support Vector Machines for Classification and Regression*, Technical Report, University of Southhampton, Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science

[19] Freund J., Wilson W. (1998): *Regression Analysis, Statistical Modeling of a Response Variable*, Academic Press, ISBN 0-12-267475-8

[20] Ong J., Abidi S. (1999): *Data Mining using Self Organizing Kohonen maps*, International Conference on AI, AI99, Las Vegas, 1999

[21] Teuvo Kohonen, Fellow, IEEE, Samuel Kaski, Member, IEEE, Krista Lagus, Jarkko Salojärvi, Jukka Honkela, Vesa Paatero, and Antti Saarela (2000): *Self Organization of a Massive Document Collection*, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 11, NO. 3, MAY 2000

[22] Tsujinishi D., Abe S. (2003): *fuzzy least squares support vector machines for multiclass problems*, Neural Networks 16 (2003), p. 785-792, www.computerscienceweb.com

[23] Suykens A.K., Gestel T.V., Branter J.D., Moor B.D., Vandewalle J. (2005): *Least Squares Support Vektor Machines*, World Scientific, ISBN 9812381511.

[24] Gupta M., Liang J., Homma N. (2003): *Static and Dynamic Neural Networks, From Fundamentals to Advanced Theory*, John Wiley & Sons, ISBN 0-471-21948-7

[25] Lai K., Yu L., Zhou L., Wang s. (2006): *Credit Risk Evaluation with Least Square Support Vector Machine*, RSKT 2006, LNAI 4092, pp. 490-495

[26] Griebel M. (2007): *Data Mining mit dünnen Gittern*, Data Mining Anwendertage, Leipzig, 2007

[27] Dahlhaus R. (2007): *Mathematical Methods in Signal Processing and Digital Image Analysis*, Springer-Verlag, ISBN 3-540-75631-0

[28] Lattner A., (2007): *Temporal Pattern Mining in Dynamic Environments*, IOS Press, ISBN 978-3-89838-309-7

Literatur: Anwendungen

[29] Lechner, Werner (1997): *In-Flight Wind Prediction Algorithm using Recurrent Neural Networks*, Neural Networks in Engineering Systems, Proceedings of the International Conference EANN 97, Stockholm Schweden,1997

[30] Lechner Werner (2003): *Intelligent Control of Autonomous Six-Legged Robots by Neural Networks*, International Federation of Automatic Control, IFAC International Conference on Intelligent Control Systems and Signal Processing, ICONS 2003, April 2003, University of Algarve

[31] Lämmel Uwe (2004): *Neuronale Netze im Data Mining: Ein Studentenprojekt* Global J. of Engng. Educ., Vol. 8, No. 3, 2004

[32] Lechner Werner (2005): *Aufbau eines mobilen Systems zur Erkennung von Gesichtern durch die optimale Kombination neuronaler Netze mit Eigenwert- und Matching-Methoden*, 10. Norddeutsche Informatik Tage, April 2005, Fachhochschule Hannover

[33] Bender T., Ehrhart S. (2007): *Finding the black nuggets - Fraud Prevention bei der 1 & 1 Internet AG*, Data Mining Anwendertage, Leipzig, 2007

[34] Schmitz M. (2007): *Data Mining bei der VPV Lebensversicherung zur optimierten Vertriebssteuerung*, Data Mining Anwendertage, Leipzig, 2007

[35] Schmitz F. (2007): *Vom Abverkauf zum Kaufanreiz - Empfehlungen durch Data Mining am Beispiel EDEKA*, Data Mining Anwendertage, Leipzig, 2007

[36] Seifert P. (2007): *Neukundengewinnung bei den Weight Watchers durch Kombination von Markt- und Kundendaten im Standortmanagement mit Hilfe der Data Mining Funktionalitäten des SQL Servers 2005*, Data Mining Anwendertage, Leipzig, 2007

[37] Witte M. (2007): *Dem Kunden einen Click voraus: Kampagnenoptimierung bei der DAB Bank AG*, Data Mining Anwendertage, Leipzig, 2007

[38] Schieder Ch. (2007): *Open Source Data Mining*, Data Mining Anwendertage, Leipzig, 2007

[39] Iversen I., Frick D. (2007): *Analytisches CRM im gesetzlichen Krankenkassen-Umfeld mit SAP CRM/BI*, Data Mining Anwendertage, Leipzig, 2007

[40] Volmer A. (2007): *Kann ein lernfähiges Computersystem Devisenkurse vorhersagen*, Modellierung eines adaptiven Prognosesystems, Swiss Finance Institute, ISBN 3-258-07335-X

[41] Kausch H. (2007): *Zeitreihenprognose: Entwurf einer Software unter Verwendung Künstlicher Neuronaler Netze*, VDM-Verlag, ISBN 3-8364-4568-9

Index